IBM

AIX 5L Porting Guide

Practical advice and guidance when porting to AIX 5L

Common problems explained and solutions documented

Written by developers for developers

Richard Cutler
Allan Cheng
Jacob Hsu
Jesper F Ljungberg
Peter Nutt
Michael Perzl

Redbooks

ibm.com/redbooks

SG24-6034-00

IBM International Technical Support Organization

AIX 5L Porting Guide

July 2001

First Edition (July 2001)

This edition applies to AIX 5L for POWER V5.1, Program Number 5765-E61, and AIX 5L for Itanium V5.1, Program Number 5799-EAR.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

When porting an application to a new platform or operating system, there are things you have to know and questions you have to ask, such as:

- What programming models are available?
- How are threads implemented?
- What link options do I need?
- Why do my makefiles not work any more?

We have tried to condense all of these questions (and answers) into one document, and this redbook is the result. It has been designed to provide guidance and reference materials for system and application programmers who have been given the task of porting applications (using the C and C++ languages) to the AIX 5L operating system. This redbook assumes the reader is familiar with the C and/or C++ programming languages and UNIX operating systems. The purpose of this book is to make your life easier.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Richard Cutler is an AIX and RS/6000 Technical Specialist at the ITSO, Austin Center. Before joining the ITSO, he worked in the RS/6000 Technical Center in the UK, where he assisted customers and independent software vendors with porting their applications to AIX.

Allan Cheng is an Advisory I/T Specialist in IBM Denmark. He has six years of experience in the IT industry. He holds a Ph.D. degree in computer science, a bachelor's degree in mathematics, and has five years experience in academic research and teaching. His areas of expertise include technical project management, consulting, AIX-based technical solution architecture (HW/SW), ERP- systems, Oracle and DB2 databases, UNIX operating systems, development, and C-programming.

Jacob Hsu is an Advisory Technical Consultant in IBM Australia. He has 10 years of experience in AIX/6000 field. He holds a master's degree in Applied Mathematical Science from the University of Georgia. His areas of expertise include AIX, RS6000, SP, Firewall, Networking (Router, Switch...), and Windows NT. He is also an MS Windows NT MCP.

Jesper F Ljungberg is an Advisory I/T Specialist in IBM Denmark. He has six years of experience in the IT industry. He has worked at IBM for three and a half years. His areas of expertise include technical project management, consulting, solution design, C programming, DB2, Oracle, AIX, and the RS/6000 hardware platform. He is *still* working on getting his master's degree in Computer Science.

Peter Nutt is a Senior I/T Specialist in IBM United Kingdom. He has recently joined IBM and spent the last 11 years in the real-time simulation and data acquisition world using C, C++, Ada, and FORTRAN. His areas of expertise cover code porting, real-time systems, and video-on-demand.

Michael Perzl is an I/T specialist from IBM Germany. He works in Pre-sales Technical Support within the Web Server Sales division of the Enterprise Systems Group of EMEA Central Region. His main responsibility is AIX technical support and porting support for ISVs and other customers. Besides AIX, he has worked with most major UNIX derivatives over the last ten years. He holds a MSc degree in Computer Science and a PhD in Mathematics, both from Munich Technical University.

Thanks to the following people for their invaluable contributions to this project:

International Technical Support Organization, Austin Center
Wade Wallace

IBM Austin
Vandana Kumar, Jim Pedersen, Ron Saint Pierre

IBM United Kingdom
Mike Pearson

Nokia Denmark
Morten Ryttov Pedersen

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in "IBM Redbooks review" on page 617 to the fax number shown on the form.
- Use the online evaluation form found at **ibm.com**/redbooks

- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. Introduction

In the middle of porting an application from one platform to another, you find yourself missing one piece of information and the man pages seem to lead you in circles - where can you go for information? This redbook has been written to help you port applications to the AIX 5L operating system by detailing the most common problems encountered, demonstrating important concepts with source code examples, and generally providing a source of reference material. It is focused on the porting process from an UNIX-based platform to AIX 5L and does *not* cover the porting of Microsoft Windows applications, as they are usually very much tied to the underlying operating system. Well-written programs that adhere to industry standards (such as POSIX.1, UNIX 95, and UNIX 98) and standard language definitions (such as ANSI C or ANSI C++) that refrain from using non-standard extensions and do not rely on platform-specific dependencies can usually be ported quite easily to a new operating system with a minimum amount of extra work besides recompiling and debugging.

In most cases, when an application is ported from a reasonably up-to-date UNIX-based operating system, the changes may be confined to becoming more compliant with industry standards or perhaps with a newer version of the same standard. Thus, after these changes are made, migration usually will require nothing more than a simple recompile. However, there are some exceptions. This redbook covers the various migration scenarios and those instances that require additional changes to the application source and/or the way the application is built. It also covers the development environment with regards to AIX 5L and the IBM C and C++ language compilers with their respective command line options.

1.1 Helpful terms and definitions

The following terms are used throughout the book. It is vital that you become familiar with the terminology used, because some terms seem very similar but have subtle yet important differences.

Source platform	Term used to describe the system (hardware and operating system) that the application is currently running on.
Target platform	Term used to describe the new operating environment that the application is being moved to. In the context of this book, the target operating system is AIX 5L.

Power systems	A term used to collectively describe systems with processors from the POWER, POWER2, PowerPC, POWER3, or RS64 families. All IBM @server pSeries and RS/6000 systems fall into this category.
Itanium-based systems	A term used to collectively describe systems with Intel® Itanium^{TM} processors.

1.2 AIX 5L benefits and features

If you are a developer working on AIX for the first time, you may be wondering, what is AIX and why should I port my code? AIX 5L is a unique and open enterprise class high-performance UNIX-based operating system that supports two different hardware platforms. AIX 5L supports both Power systems and Itanium-based systems. When your application can run under AIX 5L, you have the ability to choose the processor type of your system and the configuration. Systems range from entry level workstation/server systems to large high capacity, high availability systems with software compatibility right through the range. If you want raw power, the IBM @server range leads the field. If your application already runs on Linux, the AIX Toolbox for Linux Applications CD contains a collection of open source and GNU software built for AIX 5L for Power systems and Itanium-based systems. These tools provide the basis of the development environment of choice for many Linux application developers. All the tools are packaged using the easy to install RPM format. AIX's advanced technology and a strong Linux affinity make it the most open UNIX-based operating system in the industry.

More information on the Toolbox CD can be found in the IBM Redbook *Running Linux Applications on AIX*, SG24-6033.

Some of the main benefits and features of AIX 5L are as follows:

- Supports both 32-bit and 64-bit application environments with no performance penalty
  - Flexibility
- Supports both Power and Itanium-based systems
  - Choice
- New JFS2 file system
  - Very large file support
- WebServer enhancements

   - More caching of dynamic content
- System and network security enhancements
- Network, I/O scalability and RAS enhancements
   - Improved throughput and reliability
- Workload management tool
   - Assign system resources the way you want
- Strong Linux affinity
   - Flexibility and choice

1.3 Approaches to porting

Porting an application involves at least two platforms: one (or more) source platform(s) and a target platform. When porting an application to run on another platform, a number of factors need to be considered:

- Is anything being changed as part of the port to the target platform? For example, is the application being changed from 32-bit to 64-bit, or altered to use a different database system?
- Are all required third party packages (such as databases and class libraries) available on the target platform? Does the target platform support the same (or compatible) versions of those products?

An exercise is truly a port if the hardware and operating systems are the only things that are changed. For example, moving an application running on HP-UX with a database product to run on AIX 5L with the same version of the same database product is truly a port. However, moving the application from HP-UX and database product A to AIX 5L with either another version of database A or database product B is not just a port. In addition to making changes to the application, because the underlying operating system has changed, other changes will be required because of the change in database product. The changes are likely to be significant if a different product is used rather than just a different version of the same product. This concept is illustrated in Figure 1 on page 4. Assume that the source platform is HP-UX 10, and the application currently runs with Version 4 of database product A. The target platform is AIX 5L with Version 5 of database product A.

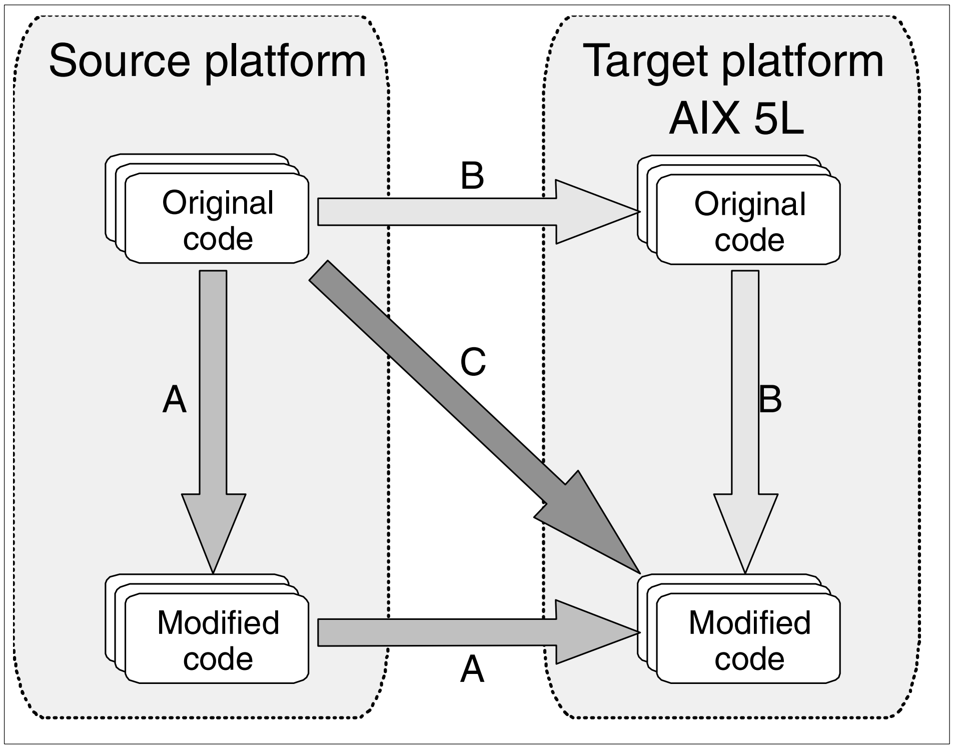


Figure 1. Porting methods

If Version 4 of database A is supported on AIX 5L, there are three possible methods that could be used to achieve the objective:

Method A The original code is modified to use Version 5 of database A while remaining on the source platform. Once this combination is proven, the code is moved to the AIX 5L system with Version 5 of database A and the port completed. Testing can be performed and the results compared against the reference source platform to prove correct operation.

Method B The code is moved to the AIX 5L system running Version 4 of database A. Testing can be performed against the reference source platform to prove correct operation. When you are satisfied that the code is functioning correctly, checkpoint the configuration. After this has been done, the operating system change (from HP-UX to AIX 5L) is complete. The database can now be upgraded to Version 5, and enhanced code can be tested against the checkpoint configuration.

Method C If approach A or B cannot be followed, for example, Version 4 of the database is not supported on AIX 5L, or Version 5 is not supported on HP-UX 10, then it will be necessary to port the code directly to use Version 5 of the database on AIX 5L.

We recommend that before you start the port, checkpoint your source code and test it to ensure correct operation. The checkpoint should include a full recompile, rebuild, and test cycle. The writers of this guide have all been involved in porting projects where time has been lost because the code did not behave correctly on the target system. Deeper investigation uncovered that the source tree used for the port included a buggy software update that was not in the original source tree. Thus, the claim often heard when porting that 'It works OK on the source system!' cannot be trusted unless you have tested it yourself.

Very often, it will not be possible to use either method A or method B because of software support issues. Using the context of the example described above, if Version 5 of database A is not available for HP-UX 10, then method A is not feasible. Similarly, if Version 4 of database A is not available for AIX 5L, then method B would not be an option. In some circumstances, method C is the only available option to move from the source platform to the target platform.

We suggest that, if possible, either method A or B shown in Figure 1 on page 4 should be used, since they have the lower risk. Method C may seem the easiest and quickest approach, but if things start to go wrong, there is more to check and no immediate reference platform that can be used to verify results. Of course, method C can work and may give you no problems at all; the choice is yours.

1.3.1 Porting steps

Within this guide, porting activity has been broken into seven main steps (see Table 1 and Figure 2 on page 7).

Table 1. Basic porting steps

Step	Name	Description
1	Prepare	Do the groundwork to enable the porting of your application. At this stage, there should be no actual coding but some research to understand the differences (if any) between the source and target platforms. Consider 32-bit and 64-bit issues, data endianness, and any implementation specific functionality that is used. Chapter 2, "Endianness - byte ordering" on page 9 and Chapter 3, "Issues regarding 32-bit and 64-bit" on page 19 provide more information.
2	Configure	Set up your development environment so that it is ready to start building your application. This is described in Chapter 4, "Setting up the development environment" on page 83. The environment includes your preferred shell, makefiles, implementation specific features, compile and link flags, and so forth.
3	Build	Chapter 5, "Porting" on page 117 through to Chapter 8, "The compilers" on page 219 contain information that may help you with compile and link problems. It is not unusual for code to go through several iterations of Steps 2 and 3 before a clean build is produced.
4	Test/debug	The code crashes - what can I do? Chapter 12, "Test and debug" on page 411 provides help with debug hints, tips, and other useful utilities.
5	Performance monitoring	Your code is running! How much CPU is it using? Can the code use more standard features? Performance tuning of applications is outside the scope of this book. AIX 5L contains a number of performance monitoring tools.
6	Enhance	The code is running correctly under AIX 5L but there may be extra performance, standards compliance, or functionality that may be obtained by code enhancement. Chapter 9, "AIX shared objects and libraries" on page 257 to Chapter 11, "C++ templates" on page 397 discusses a variety of things that may be useful to you.

Step	Name	Description
7	Build distribution pack	Will you have to support multiple systems or distribute your application to others? Refer to Chapter 20, “Packaging Software for Installation” of *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation, for information on packaging software for use with the AIX `installp` command.

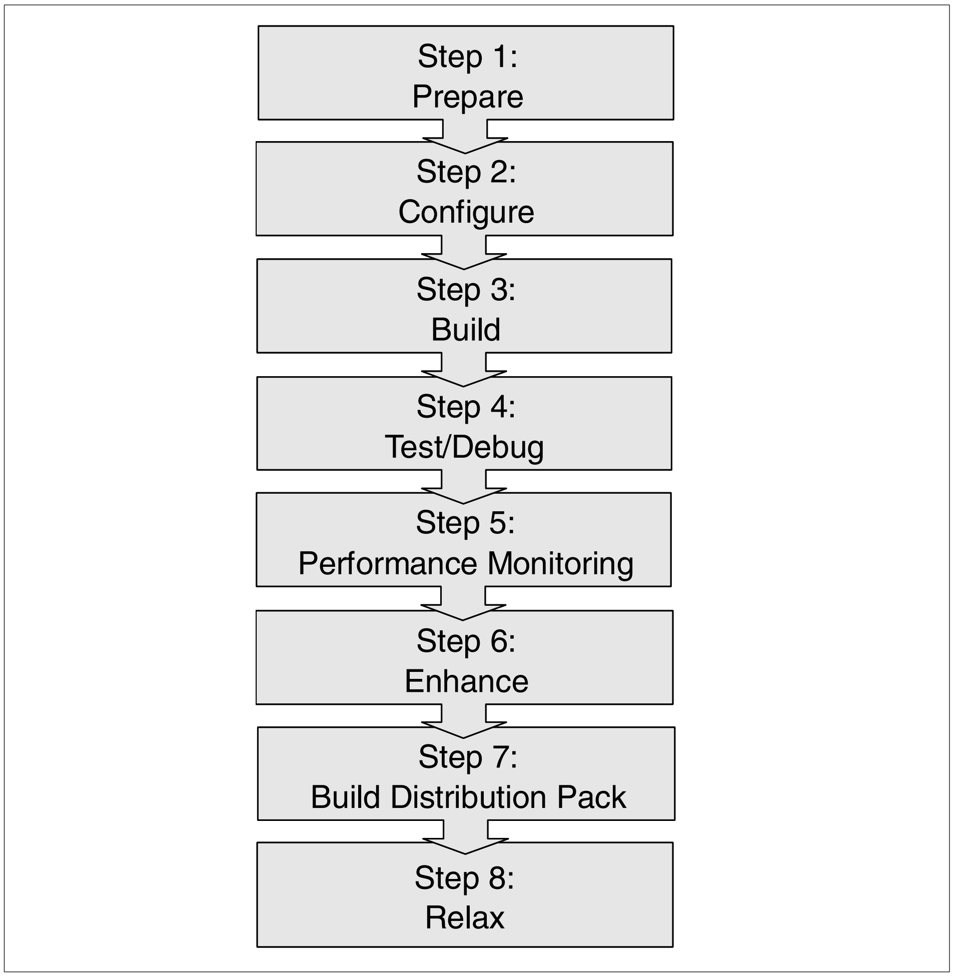


Figure 2. Porting steps

It should be noted that if the application is already running on a Linux system, you have the option to recompile it and then run it natively on AIX 5L. Many applications recompile and run without change. The AIX Toolbox for Linux Applications CD contains GNU and other commonly used tools helpful for recompiling an application for use on AIX. AIX Affinity with Linux uses a Application Programming Interface (API) approach to providing Linux application interoperability. This approach is *not* an environment or an additional operating system layer or wrapper in which to run Linux applications. It is the integration of Linux compatible APIs and header files into AIX 5L. Thus, recompiled Linux applications are native AIX applications and have access to all the reliability, scalability, and availability of AIX. The result is a tighter integration of the application to the operating system than can be achieved with an Application Binary Interface (ABI) approach. For more information, please refer to the following URL:

`http://www.ibm.com/servers/eserver/linux`

1.4 Coding practices

Much of the material presented in this book is not specific to any one vendor, nor is it specific to AIX 5L. Instead, it is just good coding practice to follow industry standards. There are many books available covering this topic, however some worthy of note are listed below.

- *The C Programming Language,* Second Edition, by Kernighan, et al
- *The Design and Evolution of C++*, by Stroustrup, et al
- *The Annotated C++ Reference Manual*, by Ellis, et al
- *C++ Primer*, Third Edition, by Lippman, et al
- *Programming Languages - C*, found at:

  `http://web.ansi.org/public/std_info.html`

  (Look for ANSI/ISO/IEC 9899-1999.)
- *Programming Languages - C++*, found at:

  `http://web.ansi.org/public/std_info.html`

  (Look for ANSI/ISO/IEC 14882-1998.)

Chapter 2. Endianness - byte ordering

This chapter covers endianness (otherwise known as byte ordering) issues and describes techniques for handling them. Byte ordering issues are often encountered by developers during the process of migrating applications, device drivers, and/or data files from one type of architecture to another.

2.1 Endianness neutrality

Although both PowerPC and Itanium architectures support big-endian (BE) and little-endian (LE) implementations, the endianness of AIX 5L running on Itanium-based systems and Power systems is different. AIX 5L for Itanium-based systems is little-endian, and AIX 5L for Power systems is big-endian. In order for an application or a device driver to use the same source code base on both platforms, it must either be endian neutral, or use conditional compilation to select appropriate code modules. A program module is considered endian neutral if it retains its functionality while being ported across platforms of different endianness. In other words, there is no relation between its functionality and the endianness of the platform it is running on.

2.1.1 Endianness - byte ordering

Endianness refers to how a data element and its individual bytes are stored and addressed in memory. Logically, in a multi-digit number, the digit with a higher order of magnitude is more significant. For example, in the four-digit number 8472, the 4 is more significant than the 7. Similarly, in multibyte numerical data, the larger the value that the byte is holding, the more significant it is. For example, the hexadecimal value, 0x123456, can be divided into three bytes: 0x12, 0x34, and 0x56 with arithmetic values of 0x120000, 0x3400, and 0x56. Obviously, byte 0x12 is the largest value; therefore, it is the most significant byte, while byte 0x56 is the smallest part and thus the least significant byte.

Most computers available today address memory in bytes while manipulating it in words (of multiple bytes). When a word is placed in memory, starting from the lowest address, there are only two sensible options: Either place the least significant byte first (known as *little-endian*) or place the most significant byte first (known as *big-endian*). The two different approaches are shown in Figure 3 on page 10.

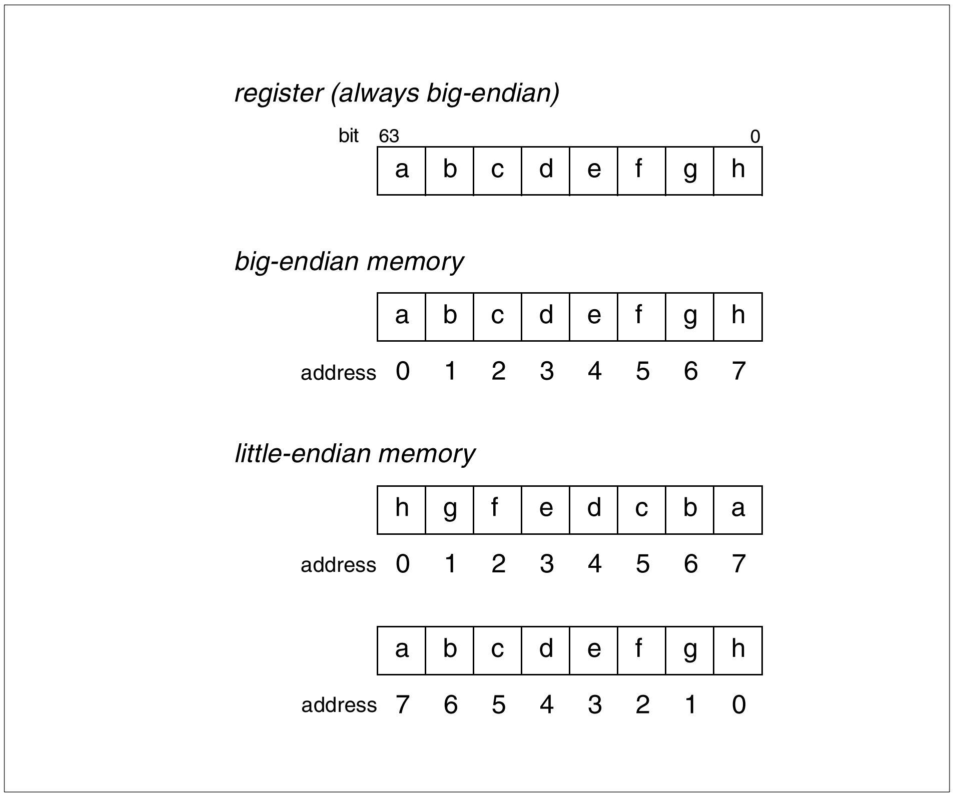


Figure 3. Byte ordering: little-endian and big-endian

In the register layout shown in Figure 3, *a* is the most significant byte, and *h* is the least significant byte. The figure also shows the byte order in memory. On big-endian systems, the most significant byte will be placed at the lowest memory address. On little-endian systems, the least significant byte will be placed at the lowest memory address.

POWER, PowerPC, most RISC-based computers, IBM 370 computers, and Internet protocol (IP) are some examples of platforms that use the big-endian data layout. Intel processors, Compaq Alpha processors, and some networking hardware are examples of systems that use the little-endian data layout.

There is an endless debate going on in the computer world about the merits of each approach; little-endian is claimed to be the most logical byte ordering while big-endian is claimed to be the most natural or intuitive one. Since a well-written application or device driver can handle both, this issue will not be discussed.

Of course, there is also the endianness of bits within one byte to consider, but it is pretty safe to assume that every system is big-endian when it comes to bit ordering in memory. In data transmission, however, it is possible to have either bit order.

2.1.1.1 Summary of general attributes of storage models

- The big-endian model addresses individual bytes in a multibyte data element from most significant byte to least significant byte (from left to right), which is similar to how data elements are referenced (from left to right).
- With the little-endian model, data elements and individual data bytes within a data element are referenced in opposite directions.
- The starting address of a data element in both storage models remains the same across the two data storage models.
- Individual bytes within a multibyte data element are addressed in a reversed order between the big-endian and little-endian data storage models.
- For single-byte data types, endianness is of no consequence; characters (or other single-byte data types) are at the same (starting) addresses in both storage models.
- The endian dependency becomes a potential problem if internal bytes and/or a proper subset of a data element are referenced individually and/or multiple data elements are referenced as an aggregated, single data element.
- Packing bit fields into a single data element can be problematic if the data needs to be stored to a persistent storage device shared by a machine using the other data storage model. But it is not an issue if the data is not shared between big-endian and little-endian machines. The internal locations of bit fields in a data element are of no consequence between the two data storage models if the bit fields are referenced as defined. Though code will work correctly (in its endianness), comments associated with code may need to be changed to present the internal bit patterns in a reversed order.

2.2 Dealing with endianness

This section describes the causes of endianness issues and recommendations for correcting them. In general, if a program module is endian neutral, the compiler will basically resolve the byte order difference between big-endian and little-endian if data is referenced consistently. That is to say:

- The multibyte data element is not referenced partially (individual byte or proper subset).
- Multiple data elements are not referenced as a single large data element.

Non-uniform data referencing is one source of endianness problem. It is often featured by data type mismatches resulting from either data element casting, use of a union data structure, or the use and manipulation of bit fields.

Sharing data across platforms is the second common source of endianness problem. For example, a big-endian system retrieves database data stored by a little-endian system.

Exchanging of data between devices of different endianness and devices on a network is the third source of endianness problems. For example, AIX on Power systems uses the big-endian model, but the PCI bus uses the little-endian model. TCP/IP protocols requires data to be sent in network byte order, which is the big-endian model.

2.2.1 General solution guideline

If you believe your code has a degree of endian sensitivity, you should:

- Identify the endianness dependency code by using `lint` or inspecting the code.
- Manually change the machine independent part of the code to make it endian neutral.
- Rewrite the machine dependent part of the code if the problems cannot be easily resolved.

2.2.2 Nonuniform data referencing

The nonuniformity in data referencing is a strength of the C language and makes the language very popular for programming system-level software, including operating systems and device drivers. The language features that enable this strength include type casting, pointer manipulation, bit fields, structures, unions, and flexible type checking. However, these very same features are also sources of endianness portability issues. For example, the C source shown in Figure 4 refers to an integer as a group of four bytes.

```
1   int main(void) {
2       int    val;
3       char   *ptr;
4
5       ptr = (char*) &val; /* pointer ‘ptr’ points to ‘val’ */
6       val = 0x89ABCDEF; /* four bytes constant */
7       printf("%X.%X.%X.%X\n", ptr[0], ptr[1], ptr[2], ptr[3]);
8       exit(0);
9   }
```

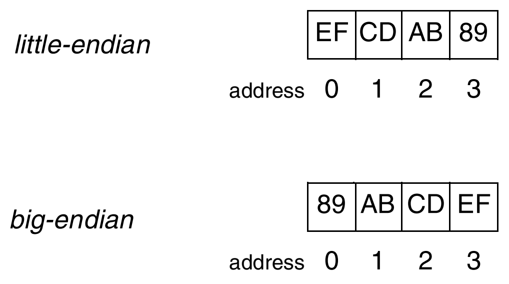


Figure 4. Nonuniform data reference using pointer

Figure 4 shows the memory layout of the 32-bit integer `val` on little-endian and big-endian systems after the assignment on line 6. We may achieve the same effect by using the union shown in Figure 5.

```
1   union {
2       int val;
3       char c[sizeof(int)];
4   } u;
5   int main(void) {
6       u.val = 0x89ABCDEF; /* four bytes constant */
7       printf("%X.%X.%X.%X\n", u.c[0], u.c[1], u.c[2], u.c[3]);
8       exit(0);
9   }
```

Figure 5. Nonuniform data reference using union

The endianness problem surfaces on line 7 where we expect to read `val` byte by byte, starting with the most significant one. As we can deduce from the figure, this program will give us EF.CD.AB.89 on little-endian platforms, rather than the 89.AB.CD.EF we may have been expecting.

2.2.2.1 Technique 1: Using macros and directives

To make the code portable, we use macros and conditional compile directives, as shown in Figure 6.

```
1   #define BIG_ENDIAN 0
2   #define LITTLE_ENDIAN 1
3   #define BYTE_ORDER BIG_ENDIAN
4   int main(void) {
5       int val;
6       char *ptr;
7       ptr = (char*) &val;
8       val = 0x89ABCDEF;
9   #if (BYTE_ORDER == BIG_ENDIAN)
10      printf("%X.%X.%X.%X\n", u.c[0], u.c[1], u.c[2], u.c[3]);
11  #else /*! BYTE_ORDER == BIG_ENDIAN*/
12      printf("%X.%X.%X.%X\n", u.c[3], u.c[2], u.c[1], u.c[0]);
13  #endif /*BYTE_ORDER == BIG_ENDIAN*/
14      exit(0);
15  }
```

Figure 6. Using macros to neutralize endianness effect

The program will be ready for little-endian platforms by making just one change on line 3: changing the definition of BYTE_ORDER from BIG_ENDIAN to LITTLE_ENDIAN.

2.2.2.2 Technique 2: Use compile time option

A better way to implement this is to define the value of BYTE_ORDER on the compiler command line. This removes the need to edit every file in a device driver or application when compiling on a new platform with a different byte order. Instead, you may only have to edit the makefiles used to build the driver or application.

2.2.2.3 Technique 3: Testing memory layout

Another approach is to test the memory layout of a predefined constant. For example, we know that the layout of a 32-bit integer variable with a value of 1 will be 00.00.00.01 for big-endian and 01.00.00.00 for little-endian. By looking at the first byte of the constant, we will be able to tell the endianness of the running platform and then take the appropriate action.

Figure 7 shows an example of determining the endianness at run time.

```
const int endian = 1;
#define is_bigendian() ( (*(char*)&endian) == 0 )

int main(void) {
    int val;
    char *ptr;
    ptr = (char*) &val;
    val = 0x89ABCDEF;
    if (is_bigendian()) {
     printf("%X.%X.%X.%X\n", u.c[0], u.c[1], u.c[2], u.c[3]);
    } else {
     printf("%X.%X.%X.%X\n", u.c[3], u.c[2], u.c[1], u.c[0]);
    }
    exit(0);
}
```

Figure 7. Determining the endianness at run time

The previous example tests the first byte of the multibyte integer, _endian_, to determine if it is 0 or 1. If it is 1, the running platform is assumed to be little-endian. Of course, the drawback to this approach is that the variable must be tested each time a data access of this type is performed, thus adding additional instructions to the code path, which of course adds a performance penalty.

The intended platform for an application or a device driver, along with the endianness of that platform, is known at compile time. Given that both device drivers and applications have performance considerations, using a compile time definition is the best method of selecting the appropriate endian-specific code segment.

2.2.3 Exchanging and sharing data

In general, these problems are typically handled by the application or data sender, which usually performs some operations on the data to convert the data to the canonical form and then sends the data. The data receiver reads the data and performs some operations to convert the data from canonical form to a usable form. In the case of the networking code, the data receiver may be either little-endian or big-endian.

2.2.3.1 Sharing data

The application programs must choose their own canonical form, decide that data will not be shared, or provide utilities to convert between the forms. XDR (eXternal Data Representation) is one of the protocols that provide a canonical data format for sharing data across heterogeneous systems.

In Figure 8, we define two macros, BE_u32() and u32_BE(), to convert a 32-bit integer from big-endian to native endian and from native endian to big-endian respectively. The macros will not do any conversion on a big-endian platform.

```
#if (BYTE_ORDER == BIG_ENDIAN)
# define BE_u32(i) (i)
# define u32_BE(i) (i)
#else /*BYTE_ORDER*/
# define BE_u32(i) ( \
            (((i)&0xFF000000) >> 24) + \
            (((i)&0x00FF0000) >> 8) + \
            (((i)&0x0000FF00) << 8) + \
            (((i)&0x000000FF) << 24) \
    )
# define u32_BE(i) ( \
            (((i)&0xFF000000) >> 24) + \
            (((i)&0x00FF0000) >> 8) + \
            (((i)&0x0000FF00) << 8) + \
            (((i)&0x000000FF) << 24) \
    )
#endif /*BYTE_ORDER*/

int main(void) {
    int val;

    val = 0x89ABCDEF;
    printf("BE(val) = %.4x\n", u32_BE(val));
    exit(0);
}
```

Figure 8. endian conversion macros

2.2.3.2 Exchanging data

One good example of exchanging data is the TCP/IP protocol, which specifies its data format in big-endian byte order. Any device driver dealing with the protocol will have to convert data from the native endianness of the running platform to big-endian before sending the data and convert it from big-endian to native-endian after receiving it from the network. This conversion can be performed by a macro that swaps data to and from a specific byte order. If the native endianness is the same as the targeted byte order, the macro will do nothing.

Network communication

In fact, in the case of TCP/IP, there are a set of conversion routines defined in POSIX to perform such operations. The routines include htonl(), ntohl(), htons() and ntohs(). The s in the routine name represents short and the l represents a 32-bit quantity. One reason that the conversion to native endianness (or host endianness) is necessary is that math operations are performed on data items, such as IP addresses (32 bits in IPv4), and TCP port values. These routines are defined as macros on AIX 5L on both Power and Itanium-based systems. The macros only perform the conversion when on little-endian Itanium-based systems. Although the macros perform no real work on Power systems, they should still be used in an application source to ensure endian neutral code.

Chapter 3. Issues regarding 32-bit and 64-bit

When preparing to port, some decisions have to be made regarding 32-bit and 64-bit programming models. This chapter is intended to shed light on related issues that should be considered upfront. We recommend that you use the AIX 5L environment that matches your source environment. In other words, avoid moving from a 32-bit to a 64-bit programming model as part of the process of migrating to AIX, because this means the exercise is no longer a true port but rather a development activity. If you wish to develop the application using the 64-bit programming model, we suggest you treat the migration to AIX as two steps (port to AIX 32-bit environment, test and verify, and then migrate to 64-bit) rather than one. This matches up with Method B, as described in Section 1.3, “Approaches to porting” on page 3, rather than Method C.

3.1 Overview of programming models

The first step in porting your source code to AIX 5L is to choose a *programming model*. The term programming model refers to the instruction set and data storage types (among other things) which are provided by the compilation tools and execution environment on a particular operating system.

3.1.1 Available programming models

AIX 5L provides you with two different programming models:

- ILP32
- LP64

3.1.1.1 ILP32

This is the native 32-bit environment for AIX 5L. Table 2 on page 28 shows the bit storage quantities for base types in this environment, where the address space is restricted to 32 bits. This programming model is the most appropriate when porting 32-bit applications to AIX 5L. More specifically, this model:

- Is appropriate for “compile-and-go" software
- Has better cache use due to smaller data sizes
- Provides automatic data conversion in and out of kernel space
- Is a fully supported environment, and not just an intermediate step during the transition to 64 bits

3.1.1.2 LP64

This is the native 64-bit environment for AIX 5L. Table 2 on page 28 shows the bit storage quantities for base types in this environment, where the theoretical address space is [0..2^{64}-1]. The LP64 programming model:

- Is the industry-wide 64-bit model
- Provides all the features (apart from data type size and alignment) of the ILP32 model
- Is appropriate for new and high-end application software

Bear in mind that 64-bit applications can only run on 64-bit hardware. Itanium-based systems and the following 64-bit Power systems are capable of running 64-bit user applications:

- RS/6000 7013 Models S70 and S7A
- RS/6000 7015 Models S70 and S7A
- RS/6000 7017 Models S70, S7A, and S80
- RS/6000 7025 Models H80 and F80
- RS/6000 7026 Models H70, H80, and M80
- RS/6000 7043 Models 260 an d270
- RS/6000 7044 Models 170 and 270
- IBM @server pSeries 680 Model S85
- IBM @server pSeries 640 Model B80
- IBM @server pSeries 660 Model 6H1
- IBM @server pSeries 620 Model 6F1

32-bit Power systems can not run 64-bit applications.

3.1.2 Porting your code

Once you have chosen the AIX 5L programming model you want your program to use, consider the issues discussed in the following sections.

3.1.2.1 Porting code to the LP64 programming model

Well-written code (see Section 1.4, “Coding practices” on page 8) that does not depend on a specific byte order or external data formats, and uses function prototypes, appropriate system header files, and system-derived data types throughout, will probably compile and run correctly in the LP64 model. This generally applies to shareware and freeware source code.

For other source code, porting to this model can be non-trivial. Among the things that might make a port challenging are:

- Code that depends on relative integer sizes (int and long)

- Code that depends on the specific size of pointers and integers
- Code that uses function calls without full prototype declarations
- Code that depends on the specific size and alignment of objects that differ between the 32-bit and 64-bit architectures
- Changes in fundamental system data types (which may have been privately defined, which is not recommended)

3.1.2.2 64-bit kernel

The 64-bit AIX 5L kernel is the *only* kernel provided on Itanium-based systems. For Power systems, both 32-bit and 64-bit kernels are provided. The 64-bit kernel can be installed and enabled on 64-bit machines.

Kernel extensions and device drivers must be compiled in 64-bit mode to be loaded into the 64-bit kernel. The 64-bit kernel provides the environment for porting and developing kernel extensions.

3.2 32-bit versus 64-bit computing

Applications continue to incorporate more and more functionality with every release and thus become more complex. As data sets grow in size, the address space requirements of these applications continue to grow. Certain classes of applications already exceed the 4 GB address space limitations of today’s 32-bit systems. Some examples of these types of applications include:

- Database applications, especially those that perform data mining
- Web caches and Web search engines
- Components of CAD/CAE simulation and modeling tools
- Scientific and technical computing applications, such as computational fluid dynamics

The primary objective for the development of 64-bit computing has been to make these and other large applications run efficiently.

Before going through the effort of converting an application from 32-bit to 64-bit, it is important to understand whether the conversion will lead to a measurable benefit in scalability and performance. To make that determination, it is important to understand the benefits of 64-bit systems and what changes to the application are needed to take advantage of these benefits.

The Intel Itanium chip is an implementation of the Intel Itanium architecture, and AIX 5L for Itanium-based systems has a 64-bit kernel. For Power systems, both 32-bit and 64-bit kernels are provided. The 64-bit kernel can be installed and enabled on 64-bit machines. This combination offers software developers the following features (not previously available on 32-bit systems)(:

- Full 64-bit addressing that expands the address space available to applications beyond the 4 GB limit on 32-bit systems
- Large process data space mapped in a large virtual address space
- Support for large data structures and executables
- Large file support using standard system library calls
- Large file caches on systems with large physical memory
- 64-bit data elements with instructions for performing efficient arithmetic and logical computations as operations, using full-register widths, the full-register set, and new instructions
- Greater scalability of system derived data types, for example, time_t and dev_t

3.2.1 Large virtual address space beyond the 4 GB barrier

The curve in Figure 9 on page 23 shows a typical performance versus problem size behavior for an application running on a machine with a large amount of physical memory installed. For very small problem sizes, the entire program can fit into the on-chip data cache and therefore runs the fastest. Only very few programs fit completely into the on-chip data cache, so for small programs the performance penalty is not that high if they still can fit into the external data cache. Even that number of programs is not very high. However, for some applications, the data area of the program becomes large enough that the program uses the entire 4 GB virtual address space available to a 32-bit application.

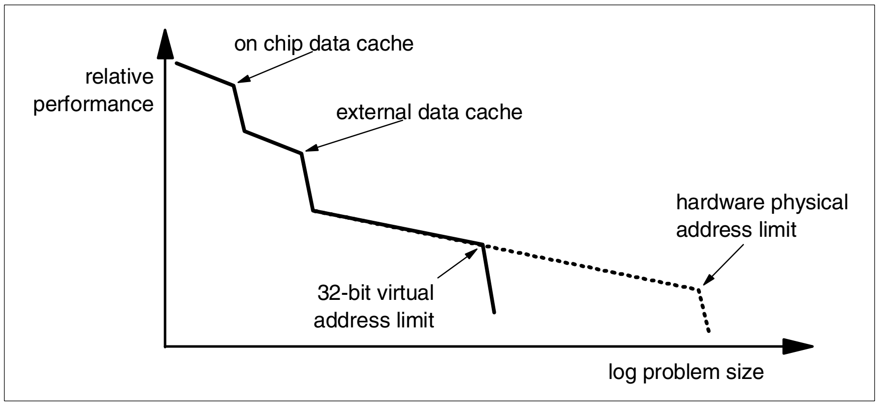


Figure 9. Typical performance versus problem size curve

Despite the 32-bit virtual address limit, 32-bit applications can still handle large problem sizes, usually by splitting the application data set between memory and disk. However, the performance penalty involved with such a step is very large, as transferring data to and from a disk drive takes orders of magnitude longer than memory-to-memory transfers.

Many servers today can easily handle more than 4 GB of physical memory, with high-end desktop machines following the same trend, but no single 32-bit program can directly address more than 4 GB at once. A 64-bit application, however, can use the 64-bit virtual address space to allow up to 18 ExaBytes (1 ExaByte is approximately 10^{18} bytes) to be directly addressed; thus, larger problems can be handled directly in memory. If the application is multithreaded and scalable, then more processors can be added to the system to speed up the application even further. These kind of applications are limited only by the amount of physical memory in the machine.

It may not seem obvious at first, but the ability to handle larger problems directly in physical memory is the most significant performance benefit of 64-bit machines. Potential examples of applications include:

- Database servers have improved performance when they can load significant portions of the database into memory.
- Larger CAD/CAE models and simulation programs may need to be able to map the entire simulation model into virtual memory.
- Larger scientific computing problems can fit into the physical memory beyond the 4 GB barrier.

- Web servers and Web caches can hold more pages in memory, thus reducing latency times.

Therefore, applications that are clearly limited by the 32-bit address space should make the transition to 64-bit mode.

3.2.2 Beyond large address space

Other reasons why you would want to make the transition of your application from a 32-bit to a 64-bit system environment include:

- Some I/O bound applications can trade off memory for disk I/O. By restructuring I/O bound applications to map larger portions of data into memory on large physical memory machines, disk I/O can be reduced. This reduction in disk I/O can improve performance because disk I/O transfers are more time-consuming than memory access.
- Many databases require data sets larger than 2 GB. It is simpler to store information for a large data set in a single file. 64-bit applications can use the standard I/O routines to access files larger than 2 GB.
- Computation with 64-bit integer quantities can be performed using the wider data paths of a 64-bit processor to gain performance.
- System interfaces have been enhanced, or limitations removed, because the underlying data types that underpin those interfaces have become larger.

3.2.3 64-bit performance considerations

When applications are compiled in 32-bit mode (ILP32) on AIX 5L, these applications usually perform better than when they are recompiled in 64-bit mode (LP64). Some of the reasons for this include:

- 64-bit programs are larger. Depending on the application, the increase in the program size can increase cache and TLB (translation lookaside buffers) misses and place greater demands on physical memory.
- 64-bit long division is more time-consuming than 32-bit integer division.
- 64-bit programs that use 32-bit signed integers as array indexes require additional instructions to perform sign extension each time an array is referenced.

Here are some ways to improve the performance of your 64-bit application:

- Avoid performing mixed 32-bit and 64-bit operations, such as adding a 32-bit data type to a 64-bit type. This operation requires the 32-bit type to be sign-extended to clear the upper 32 bits of the register.

- Avoid 64-bit long division whenever possible.
- Eliminate sign extension during array references. Change unsigned int, int, and signed int variables used as array indexes to long variables.

3.2.4 Port to 64-bit or leave your application 32-bit

Software developers must determine if the application they want to port:

- Can benefit from more than 4 GB of virtual address space, for example, for more buffer pool, for mapping files into memory, and for `shmat` and `mmap`
- Can benefit from more physical memory (greater than 4 GB) and, if so, is the user of the application likely to implement it on a system with more than 4 GB
- Needs 64-bit size integers
- Needs larger files and data structures than those supported on 32-bit systems
- Is restricted by 32–bit interface limitations
- Can benefit from full 64-bit registers to do efficient 64-bit arithmetic

The disadvantages of 64-bit applications are:

- 64-bit applications require more stack space to hold the larger registers.
- Applications have a bigger cache footprint due to the larger pointer size.
- 64-bit applications do not run on 32-bit platforms.

3.2.4.1 Estimating the effort of conversion

If the application does not necessarily need any of the features present in 64-bit operating environments, there is little reason to force a transition, especially if you want the application to be ported as quickly as possible. The application can remain as a 32-bit application and still run on a 64-bit operating system without requiring any code changes or recompilation. In fact, 32-bit applications that do not require 64-bit capabilities should probably remain 32-bit to maximize portability. As an example of this, most of the AIX 5L user commands, such as `ls`, `cat`, and `vi`, still use the 32-bit programming model, since there is no advantage in them being converted to 64-bit.

3.2.4.2 Large files support

If an application requires only support for large files, it can remain 32-bit and use the large files interface. However, if the application uses files larger than 2 GB, you might want to convert it to a 64-bit application instead of using the large file APIs directly; as in 64-bit system environments, the standard APIs can handle large files directly.

3.2.4.3 System limits removed

The 64-bit system interfaces are inherently more capable than some of their 32-bit equivalents. Application programmers concerned about year 2038 problems (when 32-bit time_t runs out of time) can use the 64-bit time_t. While the year 2038 seems a long way off, applications that do computations concerning future events, such as insurance programs or mortgages, might require the expanded time capability.

3.2.4.4 Application programming interfaces

The 32-bit application programming interfaces (APIs) supported in the 64-bit operating environment are the same as the APIs supported in the 32-bit operating environment. Thus, no changes are required for 32-bit applications between the 32-bit and 64-bit environments. However, recompiling as a 64-bit application may require cleanup of your code. See the general rules defined later in this chapter and platform specific rules in Chapter 5, “Porting” on page 117 for guidelines on how to clean up your code for 64-bit applications.

The 64-bit APIs are basically the UNIX 98 family of APIs. Their specification is written in terms of derived types. The 64-bit versions are obtained by expanding some of the derived types to 64-bit quantities. Correctly written applications using these APIs are portable in source form between 32-bit and 64-bit environments.

3.2.5 Applications requiring porting

Applications with the following characteristics will almost certainly need changes when being ported from 32-bit to 64-bit system environments:

- Read and interpret kernel memory directly
- Use the /proc file system to access 64-bit processes
- Use a library that has only a 64-bit version
- Device drivers
- Interoperability issues

3.3 Migrating from 32-bit to 64-bit

Most applications are written in one or more high-level languages. Since applications written in assembly or macro assembly will need complete rewrites, porting issues will generally be discussed in terms of the C language. Variations of the problems occur with other languages too.

Most well-written programs will compile and run without change, where "well-written" implies the use of good programming practices, including:

- Conformance to the ANSI/ISO C standard
- Portability considerations high in the design, implementation, and maintenance phases of the software cycle
- Use of prototyped functions declarations throughout

In reality, however, portability issues are often the first to be sacrificed to meet completion schedules and performance issues or are forgotten in the software maintenance cycle. In other cases, production code may also be written without regard for portability. The base of much of the source code to be ported to AIX 5L has existed for a long time and been through many maintenance and enhancement cycles. Therefore, it may have been quite easy during these processes to assume, implicitly or explicitly, either absolute or relative sizes of integers (int), long integers (long), and pointer data types to become part of the source code. These assumptions, in combination with the changes in size and alignment of basic data types, are the source of most problems porting existing source code to a 64-bit system.

This section details the areas where problems may occur and explains how the C compiler, in combination with `lint`, can be used to isolate and correct problems. Most solutions to the problems described here are not tied specifically to AIX 5L but are generally applicable when migrating from 32-bit to 64-bit environments.

3.4 Conversion of 32-bit applications

Application developers have to deal with two basic issues when converting their 32-bit applications into 64-bit applications:

- Data type consistency and the different data model
- Interoperation between applications using different data models

It is usually better to maintain a single source with as few `#ifdefs` as possible than to maintain multiple source trees. Therefore, it is best to write code that works correctly in both 32-bit and 64-bit environments. At best, the conversion of current code might require only a recompilation and relinking with the 64-bit libraries. For those cases where code changes are required, AIX 5L includes tools that help make the transition easier.

3.4.1 ILP32 and LP64 programming models

The ANSI/ISO C standard specifies that C must support four signed and four unsigned integer data types, namely char, short, int, and long. Unfortunately, there are only a few requirements imposed by the ANSI standard on the sizes of these data types. According to the standard, int and short should be at least 16 bits, and long should be at least as long as int, but not smaller than 32 bits.

A different data type model is used in the 64-bit environment than in the 32-bit environment. The C and C++ data type model used for 32-bit applications is the ILP32 model, so named because integers (int), long integers (long), and pointers are 32 bits. The LP64 data model is the C and C++ data type model for 64-bit applications. This programming model was agreed upon by a consortium of companies across the industry. It is so named because long integers (long) and pointers in this data model are 64-bit quantities. The standard relationship between C and C++ integral data types still holds true:

```
sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
```

The LP64 data model is the emerging standard on 64-bit UNIX-based systems provided by all major system vendors. Applications that transition to the LP64 data model are therefore highly portable to other LP64 vendor platforms.

Table 2 lists the basic C and C++ data types and their corresponding sizes in bits for both the ILP32 and LP64 programming models.

Table 2. Basic C and C++ type sizes in bits in the ILP32 and LP64 model

C and C++ data type	ILP32	LP64
char	8	unchanged
short	16	unchanged
int	32	unchanged
long	32	64
long long	64	unchanged
pointer	32	64
enum	32	unchanged
float	32	unchanged
double	64	unchanged
long double	64	unchanged

Using appropriate compiler options, the size of long double can be increased to 128 bits for both ILP32 and LP64 models on Power systems.

Some fundamental changes occur when an application is moved from the ILP32 programming model to the LP64 programming model:

- Long integers (long) and integers (int) are no longer the same size.
- Pointers and integers (int) are no longer the same size.
- Pointers and long integers (long) are 64 bits and are aligned on 64-bit boundaries.
- Certain predefined types, such as size_t and ptrdiff_t are 64-bit integral types.

These differences can potentially have a big impact during the porting process in the following areas:

- Data truncation
- Pointer assignment and arithmetic
- Constants
- Bit shifts and bit masks
- Bit fields
- Enumerated types
- Data alignment and data sharing
- Casting
- Lack of function prototypes
- Data type promotion

When porting 32-bit applications to 64-bit environments, most of the problems that will be encountered due to changed data type sizes are:

- Different sizes for int (32 bits) and long (64 bits) and interchangeable use of them
- Different sizes for int (32 bits) and pointers (64 bits) and interchangeable use of them
- Due to different data type sizes, objects (for example, structs) may change their size and alignment

It is most likely that most code you end up changing incorrectly assumes the following relation:

```
sizeof(int) == sizeof(void *) == sizeof(long)
```

3.4.2 32-bit and 64-bit application interoperability

Some restrictions apply when objects, such as data and memory, are shared between 32-bit and 64-bit applications. These restrictions also apply when objects are shared between 32-bit applications and the 64-bit version of the operating system.

3.4.2.1 Same size and alignment

If data is shared between 32-bit and 64-bit applications, then all data items must have the same size and alignment within both applications.

3.4.2.2 Shared memory

32-bit applications can only attach to shared memory segments which exist in 32-bit virtual address space.

3.4.2.3 Message queues

The size of a message queue is defined as type size_t. On a 32-bit system size_t has a size of four bytes (32 bits) while on a 64-bit system it is eight bytes (64 bits). If the 64-bit application wants to exchange data with 32-bit applications using message queues, it has to make sure that the size of the message queue does not exceed the largest 32-bit unsigned value (2^{32}-1).

3.4.2.4 Memory-mapped files

Memory-mapped files that should be shared between 32-bit applications have to be mapped into a 32-bit virtual address space.

3.4.2.5 Symbols

Symbols within 64-bit executables of AIX 5L are assigned 64-bit values. If an application wants to extract 64-bit values from the symbol table of a 64-bit executable, it needs 64-bit data fields. The nlist64 subroutine runs in both 32-bit and 64-bit mode and can read both 32-bit and 64-bit files in both 32-bit and 64-bit modes.

3.4.2.6 Large files

32-bit applications can open, create, and use large files with the large file enabled programming environment. A large file is a file that is 2 GB or greater in size. However, when creating or opening large files, the 32-bit application must specify the O_LARGEFILE flag with the `open` system call or use the `open64` system call.

The use of lseek within a 32-bit application to position a file pointer is limited to the 2 GB mark. To position the file pointer beyond that mark the lseek64 subroutine should be used.

3.5 ANSI C integer conversion rules

To better understand the occurrence of possible conversion problems when migrating from ILP32 mode to LP64 mode, it helps to understand the integer conversion rules for ANSI C. The conversion rules that seem to cause the most problems between 32-bit and 64-bit integral values are the following (we assume here that negative integer numbers are represented by the twos-complement):

- Integral promotion
  - The general rule for converting from one integer type to another is that the mathematical value of the result should equal the original mathematical value if that is possible. For example, if an unsigned integer has the value 20 and this value is to be converted to a signed type, the resulting value should be 20 also.
  - If it is not possible to represent the original value of an object of the new type, then there are two cases. If the result type is a signed type, then the conversion is considered to have overflowed and the result value is technically not defined. If the result type is an unsigned type, then the result must be that unique value of the result type that is equal (congruent) mod 2^n to the original value, where n is equal to the number of bits used in the representation of the result type. If signed integers are represented using twos-complement notation, then no change of representation is necessary when converting between signed and unsigned integers of the same size.
- Conversion between signed and unsigned integers
  - When an unsigned integer is converted to a signed integer of the same size, the conversion is considered to overflow if the original value is too large to be represented exactly in the signed representation (that is, if the high-order bit of the unsigned number is 1). However, many programmers and many programs depend on the conversion being performed quietly and with no change of representation to produce a negative number.
  - If the destination type is longer than the source type, then the only case in which the source value will not be representable in the result type is when a negative signed is converted to a longer, unsigned type. In that case, the conversion must necessarily behave as if the source value were first converted to a longer signed type of the same size as the destination type, and then converted to the destination type.
  - If the destination type is shorter then the source type, and both the original type and the destination type are unsigned, the conversion can

be effected simply by discarding high-order bits from the original value; the bit pattern of the result representation will be equal to the *n* low-order bits of the original representation. This same rule of discarding works for converting signed integers in twos-complement from to a shorter unsigned type. Note that this rule will not preserve the sign of the value in case of overflow, but the action on overflow is not defined anyway.

For a more detailed discussion of the conversion rules, refer to the ANSI C standard and see Section 1.4, "Coding practices" on page 8.

3.6 C and C++ data type size issues

This section describes the C and C++ language data types in AIX 5L and differences in *size* between the data types in 32-bit and 64-bit programming models. It also describes the porting issues that may be encountered and methods that can be used to write programs so that they are not impacted by those differences.

3.6.1 C and C++ data type sizes in AIX 5L

Support for a 64-bit address space and larger scalar arithmetic ranges in LP64 mode naturally requires changes in at least some of the basic C and C++ data types. Details of the size characteristics (in bytes) of C and C++ language base data types in each programming model are shown in Table 3.

Table 3. Comparison of C and C++ type sizes in bytes for AIX 5L

	AIX 5L for Power systems		**AIX 5L for Itanium-based systems**	
Data type	**ILP32 Model**	**LP64 Model**	**ILP32 Model**	**LP64 Model**
char	1	1	1	1
short	2	2	2	2
int	4	4	4	4
long	4	8	4	8
long long	8	8	8	8
float	4	4	4	4
double	8	8	8	8
long double	8 / 16	8 / 16	8 / 16	8 / 16

	AIX 5L for Power systems		AIX 5L for Itanium-based systems	
Data type	ILP32 Model	LP64 Model	ILP32 Model	LP64 Model
pointer	4	8	4	8

The differences between the LP64 programming model and the others are the size of long, long long, pointer, and long double data types. As shown in Table 3 on page 32, pointers and long integers in LP64 mode are eight bytes (64 bits) while in a 32-bit application context they are only four bytes (32 bits). The size of long double floating point types has changed for performance considerations.

Although the 32-bit Power model supports 16-byte (128 bits) long double variables with 128-bit alignment the default size for long double in Power is 8 bytes (64 bits) when the -qlongdouble or -qldbl128 option is used with the compiler.

In Section 5.2, “System derived data types” on page 120, commonly derived data types already defined by the operating system, such as time_t are explained in further detail.

The cardinality of data types which have the same byte sizes in ILP32 and LP64 mode is shown in Table 4.

Table 4. Cardinality of types with the same size in ILP32 and LP64 mode

	ILP32 mode & LP64 mode	
Data type	min. value	max. value
signed char	-128	127
unsigned char	0	255
signed short	-32,768	32,767
unsigned short	0	65,535
int	-2,147,483,648	2,147,483,647
unsigned int	0	4,294,967,295

The cardinality of data types which have different sizes in ILP32 and LP64 mode is shown in Table 5.

Table 5. Cardinality of types with different sizes in ILP32 and LP64

	ILP32 mode		LP64 mode	
Data type	**min. value**	**max. value**	**min. value**	**max. value**
signed long	-2,147,483,648	2,147,483,647	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long	0	4,294,967,295	0	18,446,744,073,709,551,615
void *	0	4,294,967,295	0	18,446,744,073,709,551,615

The system header file <limits.h> defines the following integer constants that should be used instead of the literal constants above, as seen in Table 6.

Table 6. Integer constants defined by the system header file <limits.h>

Constant	Description	Numeric value
CHAR_BIT	number of bits per char	8
SCHAR_MAX	biggest signed char	127
SCHAR_MIN	smallest signed char	-SCHAR_MAX-1
UCHAR_MAX	biggest unsigned char	255
SHRT_MAX	biggest signed short	32,767
SHRT_MIN	smallest signed short	-SHRT_MAX-1
USHRT_MAX	biggest unsigned short	65,535
INT_MAX	biggest signed int	2,147,483,647
INT_MIN	smallest signed int	-INT_MAX-1
UINT_MAX	biggest unsigned int	4,294,967,295
LONG_MAX	biggest signed long	INT_MAX or 9,223,372,036,854,775,807
LONG_MIN	smallest signed long	INT_MIN or -LONG_MAX-1

Constant	Description	Numeric value
ULONG_MAX	biggest unsigned long	UINT_MAX or 18,446,744,073,709,551,615

Note that the value of LONG_MAX, LONG_MIN and ULONG_MAX depends on the programming model in use (ILP32 or LP64).

The system header file <float.h> defines the float constants (shown in Table 7) that should be used.

Table 7. Floating point constants defined by the system header file <float.h>

Constant	Description	Numeric value
FLT_RADIX	Exponent radix	2
FLT_MANT_DIG	Number of bits in the significand	24
FLT_EPSILON	1ulp when exponent = 0	1.1920928955078125e-7
FLT_DIG	Number of decimal digits of precision	6
FLT_MIN_EXP	Exponent of smallest NORMALIZED float number	-125
FLT_MIN	Smallest NORMALIZED float number	1.1754943508222875e-38
FLT_MIN_10_EXP	Minimum base 10 exponent of NORMALIZED float	-37
FLT_MAX_EXP	Exponent of largest NORMALIZED float number	128
FLT_MAX	Largest NORMALIZED float number	3.4028234663852886e+38
FLT_MAX_10_EXP	Largest base 10 exponent of NORMALIZED float	38
DBL_MANT_DIG	Number of bits in the significand	53
DBL_EPSILON	1ulp when exponent = 0	2.2204460492503131e-16
DBL_DIG	Number of decimal digits of precision	15

Constant	Description	Numeric value
DBL_MIN_EXP	Exponent of smallest NORMALIZED double number	-1021
DBL_MIN	Smallest NORMALIZED double number	2.2250738585072014e-308
DBL_MIN_10_EXP	Minimum base 10 exponent of NORMALIZED double	-307
DBL_MAX_EXP	Exponent of largest NORMALIZED double number	1024
DBL_MAX	Largest NORMALIZED double number	1.7976931348623158e+308
DBL_MAX_10_ EXP	Largest base 10 exponent of NORMALIZED double	308

Since the default size for long double on Power systems is 8 bytes (64 bits), the same as for double, the values of the constants in Table 8 only change to the shown values when using the -qlongdouble or -qldbl128 compiler options.

Table 8. Long double constants for Power systems

Constant	Description	Numeric value
LDBL_MANT_DIG	Number of bits in the significand	106
LDBL_EPSILON	1ulp when unbiased exponent = 0	0.24651903288156618919116517665087070E-31
LDBL_DIG	Number of decimal digits of precision	31
LDBL_MIN_EXP	Exponent of smallest NORMALIZED long double number	DBL_MIN_EXP
LDBL_MIN	Smallest NORMALIZED long double number	((long double) DBL_MIN)
LDBL_MIN_10_EX	Minimum base 10 exponent of NORMALIZED long double	DBL_MIN_10_EXP

Constant	Description	Numeric value
LDBL_MAX_EXP	Exponent of largest NORMALIZED long double number	DBL_MAX_EXP
LDBL_MAX	Largest NORMALIZED long double number	0.1797693134862315807937289714053023E+309
LDBL_MAX_10_EXP	Largest base 10 exponent of NORMALIZED long double	DBL_MAX_10_EXP

For Itanium-based systems, the values for long double use 80 of the 128 bits and are shown in Table 9. For 64-bit long double, obtained using the -qlongdouble=64 compiler option, the values are as for double.

Table 9. Long double constants for Itanium-based systems

Constant	Description	Numeric value
LDBL_MANT_DIG	Number of bits in the significand	64
LDBL_EPSILON	1ulp when unbiased exponent = 0	1.0842021724855044340075E-19
LDBL_DIG	Number of decimal digits of precision	18
LDBL_MIN_EXP	Exponent of smallest NORMALIZED long double number	-16381
LDBL_MIN	Smallest NORMALIZED long double number	3.36210314311209350626e-4932
LDBL_MIN_10_EX	Minimum base 10 exponent of NORMALIZED long double	-4931
LDBL_MAX_EXP	Exponent of largest NORMALIZED long double number	16384
LDBL_MAX	Largest NORMALIZED long double number	1.18973149535723176502e+4932
LDBL_MAX_10_EXP	Largest base 10 exponent of NORMALIZED long double	4932

3.6.2 Different sizes for int and long in LP64 mode

In ILP32 mode, both int and long data types are 32 bits in size. Because of this similarity, these types may have been used interchangeably in production code. As shown in Table 3 on page 32, in LP64 mode, the data type long is 64 bits in length. A general guideline is to review the existing use of long data types throughout the source code. If the values to be held in such variables, fields, and parameters will fit in the range of [-231...231-1] or [0...232-1], then it is probably best to use int or unsigned int, respectively.

Obviously, the size difference of data types means that applications will require changes when being ported from a 32-bit to a 64-bit platform, particularly when related to data structures. For example, a 32-bit application is expecting to read the following data structure from memory located on a device controller:

```
struct {
    long   length;  /* buffer length field */
    char * buffer;  /* pointer to memory on the controller */
    int    flags;   /* miscellaneous flags */
} memio_t;
```

All three variables are stored on the device controller as 32-bit values. This example only considers the issue of data type size. When reading data from a device controller, the endianness of the memory being read and the endianness of the device driver performing the read operation must also be considered. See Chapter 2, “Endianness - byte ordering” on page 9, for a discussion of endian issues.

Figure 10 shows the layout of the data structure in both 32-bit and 64-bit models.

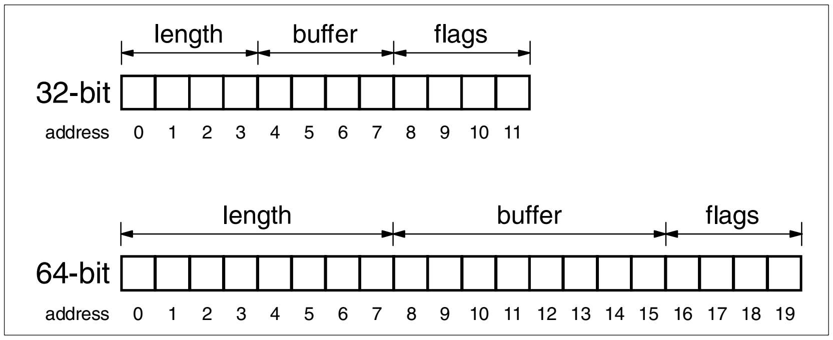

length
buffer
flags
32-bit
address 0 1 2 3 4 5 6 7 8 9 10 11
length
buffer
flags
64-bit
address 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Figure 10. Layout of sample data structure on 32-bit and 64-bit platforms

Only one of the structures corresponds to the data read from the device controller. Since the device driver using this definition works correctly in a 32-bit ILP32 model, if the same structure is used on a 64-bit LP64 model, the driver will not work correctly. This is because the size of a long differs between the two environments. Instead, the structure must be defined using types of the correct size in the particular compilation environment.

To yield the same result as on a 32-bit model, the code must be guarded with a preprocessor directive to produce a conditional compile section of code. The compiler will then use the correct definition for the compilation model. The result will be similar to the following:

```
struct {
#ifdef __64BIT__
    int32_t length;  /* buffer length field */
    __ptr32 buffer;  /* pointer to memory on the controller */
#else /*__64BIT__*/
    long    length;  /* buffer length field */
    char*   buffer;  /* pointer to memory on the controller */
#endif /*__64BIT__*/
    int     flags;   /* miscellaneous flags */
} memio_t
```

See Section 5.2, “System derived data types” on page 120 for commonly derived data types already defined by the operating system. The use of these data types (specified by the ANSI C standard) guarantees a specific size of a data type on all platforms.

3.6.3 The sizeof() operator

The sizeof() operator is used to obtain the size of a type or data object. The result is a constant integer value. In ANSI C, the result of sizeof() has the unsigned integer type size_t defined in the header file <stddef.h>. Traditional C implementations often use int or long as the result type.

In LP64 mode, the sizeof() operator has the effective data type of unsigned long. If the result of sizeof() is passed to a function expecting an argument of type int, or assigned or cast to an int, this truncation might cause the loss of data in some cases. Consider the code fragment below:

```
long a[50];
unsigned char size = sizeof(a);
```

In ILP32 mode the size of the array a is 200 bytes (as longs are 4 bytes in size). However, in LP64 mode, the size of the array a is 400 bytes (as longs are 8 bytes in size).

3.6.3.1 Recommendation

It is recommended that you check throughout your code to see that all variables have the appropriate cardinality for the range of possible values, so that no undesired side-effects can occur.

3.6.3.2 lint assistance

Unfortunately, neither the VisualAge C compiler nor `lint` help you find occurrences of this type of error. However, the GNU C Compiler produces the following warning when invoked with the -Wall option:

```
.... warning: large integer implicitly truncated to unsigned type
```

3.6.4 Data type specifications in (s)printf/(s)scanf

In source code written for 32-bit environments, variables of type int, long, and pointer are often interchanged with no impact on the application, since the types are all the same size. When moving code to the 64-bit environment, the difference in size can cause problems. Even simple things, such as calls to the printf or scanf family of routines, are affected. Consider the following code example:

```
struct devinfo_t { .... } devinfo;

(void) printf("addr(devinfo) = %x\n", (void *) &devinfo);
```

3.6.4.1 Recommendation

Consider the following recommendations when using format specifiers with the printf()/scanf() family of routines:

- The format strings for printf, sprintf, scanf, and sscanf have to be changed for long arguments. The long size specification, l (lower case L), should be prepended to the conversion operation character in the format string to identify a long integer data type. For example:

  ```
  printf("%ld\n", 7FFFFFFFL);
  ```

- The format strings for printf, sprintf, scanf, and sscanf have to be changed for pointer arguments. The conversion operation specified in the format string should be changed from %x to %p in order to work in both the 32-bit and 64-bit environments. For example:

  ```
  printf("%p\n", argv[0]);
  ```

  This prints the value of the pointer in hexadecimal format.

- Do not use a hardcoded field width to format pointers. For example, the following is used to print pointers zero padded to eight digit hexadecimal digits:

  ```
  printf("0x%08p\n", argv[0]);
  ```

If you want to ensure that the hexadecimal value is zero padded to the appropriate value for the execution mode, use the `*` (asterisk) format option. The following example will pad to eight hexadecimal digits in 32-bit mode and 16 hexadecimal digits in 64-bit mode:

```
printf("0x%0*p\n", (int) (2*sizeof(argv[0])), argv[0]);
```

- Macros for specifying the (s)printf and (s)scanf format specifiers are provided in the ANSI C header file <inttypes.h>. These macros prepend the format specifier with an l or ll to specify the argument as a long or long long type, given that the number of bits in the argument is built into the name of the macro.
  - Macros for (s)printf format specifiers exist for printing 8-bit, 16-bit, 32-bit, and 64-bit integers, the smallest integer types, and the largest integer types, in decimal, octal, unsigned, and hexadecimal form. For example:

    ```
    int64_t i;
    printf("i = %" PRIx64 "\n", i);
    ```

  - Macros for (s)scanf format specifiers exist for reading 8-bit, 16-bit, 32-bit, and 64-bit integers, and the largest integer type in decimal, octal, unsigned, and hexadecimal form.

    ```
    uint64_t u;
    scanf("%" SCNu64 "\n", &u);
    ```

3.6.4.2 lint assistance

Unfortunately neither the VisualAge C compiler or `lint` help you find occurrences of this type of error. However, the GNU C Compiler produces the following warning when invoked with the -Wall option for the code example above:

```
.... warning: unsigned int format, pointer arg (arg 1)
```

3.6.5 Structures and unions may change size

The 64-bit environment can affect the size of structures and unions. Because pointers and long integers (long) are 64-bit values in LP64 mode, structures and unions that include pointers or long data types are bigger in size than the same structures in 32-bit mode.

3.6.5.1 Structures

Consider the following code example, which describes a simple double-linked list:

```
struct dlinklist_t {
   char *text;
```

```
   int value;
   struct dlinklist_t *prev;
   struct dlinklist_t *next;
};
```

The size of this structure in ILP32 mode is 16 bytes (4+4+4+4 bytes) while in LP64 mode the sum of all member sizes is 28 bytes (8+4+8+8 bytes) but due to alignment issues (see Section 3.10, “C and C++ data type alignment issues” on page 65), the size of struct dlinklist_t in LP64 mode is 32 bytes.

3.6.5.2 Unions

Problems arise in LP64 mode when the use of unions is based on assumptions such as:

```
sizeof(double) == 2 * sizeof(long) or sizeof(long) == 4 * sizeof(char)
```

Consider the following code example that wrongly assumes that an array of two long integers overlays a double:

```
union double_overlay_wrong {
   double d;
   unsigned long x[2];
}
```

The correct way to do it in LP64 mode would be to change the long to an int:

```
union double_overlay_correct {
   double d;
   unsigned int x[2];
}
```

3.7 Data truncation

Truncation problems can arise when assignments are made between 64-bit and 32-bit data items. Since integers (int), long integers (long), and pointers are 32 bits in ILP32, a mixed assignment between these data types did not represent any problem. It does, however, in LP64 mode, as long integers (long) and pointers are no longer the same size as integers (int). In LP64 mode, data truncation can occur during:

- Assignment of long to a smaller type
- Assignment of long to double
- Integer expression with potential overflow is converted to a long
- Explicit cast is improperly applied

3.7.1 Assignment of long to a smaller type

The assignment of a long integer value to a smaller type will result in truncation of the 64-bit value. See Section 3.5, “ANSI C integer conversion rules” on page 31 for the rules used in this conversion. This truncation may have been exactly the intention of the code, but where int and long variables have been used interchangeably while implicitly or explicitly assuming that they are interchangeable, this may be an unexpected source of problems. For example, consider the following code example shown in Figure 11.

```
01  extern long dosomething(int);
02
03  int main(int argc, char *argv[])
04  {
05    int i1, i2, i3;
06    long l1, l2, l3;
07
08  /* implicit truncation occurs in the next 3 statements */
09    i1 = l1;
10    i2 = i2 * l2;
11    i3 = dosomething(l3);
12
13  /* use explicit casting to obtain the intended narrowing */
14    i1 = (int) l1;
15    i2 = (int) i2 * l2;
16    i3 = (int) dosomething((int) l3);
17  }
```

Figure 11. Data truncation during assignment

3.7.1.1 Recommendation

Examine all instances of narrowing assignment, particularly where the source is of type long or unsigned long, and decide whether such narrowing may be a problem or is intended. Use an explicit cast at the point of an intended narrowing conversion (lines 14 to 16) to indicate to the compiler and to `lint` that such narrowing is being done by design.

3.7.1.2 lint assistance

`lint` will report implicit narrowing integral conversions associated with the assignment operator (`=`), as shown in the following `lint` output:

```
"truncate.c", line 9: warning: conversion from long may lose accuracy
"truncate.c", line 10: warning: conversion from long may lose accuracy
"truncate.c", line 11: warning: mismatched type in function argument
"truncate.c", line 11: warning: conversion from long may lose accuracy
```

```
"truncate.c", line 11: warning: conversion from long may lose accuracy
"truncate.c", line 14: warning: conversion from long may lose accuracy
"truncate.c", line 15: warning: conversion from long may lose accuracy
"truncate.c", line 16: warning: conversion from long may lose accuracy
"truncate.c", line 16: warning: conversion from long may lose accuracy
```

The reason for the warnings above is that

- A long value is assigned to an int value (lines 9, 10, 11, 14, 15, 16).
- The expressions in line 10 and 15 are of type long as i2 is promoted from type int to type long before the multiplication.
- In line 11, a long value is passed as a parameter to the function dosomething, therefore possibly truncating the long value according to the ANSI C integer conversion rules.

3.7.2 Assignment of long to double

On 32-bit systems, the code can safely assume that the data type double can contain an exact representation of any value stored in a long or pointer data type. In ILP32 mode, long is 32 bits and double is 64 bits, with a mantissa of 53 bits.

The standard double floating-point representation is given by the *IEEE Standard for Binary Floating-Point Arithmetic* (ANSI/IEEE Std 754, 1985) as:

$$x_{double} = s \times 2^{e} \times \sum_{k=1}^{53} f_k \times 2^{-k} \qquad -1021 \le e \le 1024$$

where:

s is the sign (± 1)
e is the exponent value
f_k are the significand digits, $0 \le f_k < 2$

When converting code which makes this assumption (from ILP32 mode to LP64 mode), this assumption is no longer valid, as longs and pointers grow to 64 bits. For example, the code example shown in Figure 12 on page 45 behaves differently in 32-bit and 64-bit mode.

```
01  #include <stdio.h>
02
03  int main(int argc, char *argv[])
04  {
05    long l1 = 111111111111111111L;
06    long l2;
07    double d;
08
09    d = l1;
10    l2 = d;
11
12    if (l1 == l2)
13      printf("ILP32 mode: long = double is OK\n");
14    else
15      printf("LP64 mode: long = double is NOT OK\n");
16  }
```

Figure 12. Code example longdouble.c

When compiled as a 32-bit program (in ILP32 mode), the value of l1 can be assigned to the double d without loss of precision:

```
% xlc -q32 -o longdouble32 longdouble.c
% ./longdouble32
ILP32 mode: long = double is OK
%
```

However, when compiled as a 64-bit program (in LP64 mode), the value of l1 is truncated when assigned to the double d and therefore the condition in line 12 can never be true:

```
% xlc -q64 -o longdouble64 longdouble.c
% ./longdouble64
LP64 mode: long = double is NOT OK
%
```

3.7.2.1 Recommendation

Examine all assignments of a long to a double, as they will most likely result in a loss in accuracy. If this is the case, you may try using long double, although, ideally, code should be written so that data casting of this type is not required.

3.7.2.2 Compiler assistance

The -qwarn64 option of the VisualAge C compiler can help you by finding occurrences where a variable of type long might overflow in ILP32 mode and therefore suffer precision loss if being assigned to a variable of type double. Here is the relevant output:

```
line 5.13: 64-bit portability: constant which will overflow in 32-bit mode
may select unsigned long int or long int in 64-bit mode
```

3.7.3 Integer expression with potential overflow

Arithmetic expressions are evaluated following the usual arithmetic conversions of all operands to a common type as defined by the ANSI C integer conversion rules. The type of the expression is this common type. For an expression containing integral operands, that implies that small operands will be converted to int as needed, to represent all values of the original type. The common type will only be larger than an int if an operand of the expression is an unsigned int, long, or unsigned long. What this means for LP64 mode is that expressions not containing a long or unsigned long type will be evaluated in terms of 32-bit values and yield a 32-bit result.

The example in Figure 13 illustrates an instance where, in LP64 mode, the actual results may not be what the original intention might have been.

```
01  int x, y;
02  long l;
03
04  void dosomething(void)
05  {
06    l =         x * y;   /* 32-bit multiply, potential truncation */
07    l = (long) (x * y);  /* 32-bit multiply, potential truncation */
08
09    l = (long) x * y;         /* 64-bit multiply, no truncation */
10    l =        x * (long) y;  /* 64-bit multiply, no truncation */
11  }
```

Figure 13. Potential data truncation

If the user intended to get a 64-bit result from the multiplication, lines 6 and 7 are incorrect. Both source statements contain a 32-bit multiplication with the result truncated to 32 bits and cast, implicitly in line 6 and explicitly in line 7, to a 64-bit long value. Lines 9 and 10 show the correct way to get a 64-bit multiplication of two smaller types. By casting either operand to a long prior to the multiplication, the other operand is implicitly cast to a long.

3.7.3.1 Recommendation

In order to have integral expressions produce results which are 64 bits long, at least one of the operands must have a data type of long or unsigned long.

Therefore, if none of the operands involved is of type long, an appropriate cast to long or unsigned long should be applied to one of the operands. That cast will have the effect of percolating up the expression tree to yield a 64-bit result.

3.7.3.2 lint assistance

In the case demonstrated in Figure 13 on page 46, `lint` is not able to assist you, as it would have to guess what your original intention might have been. It is therefore recommended that you check throughout your code for these cases. A cast applied to an expression of int or unsigned int type, implicitly or explicitly, will only yield a 64-bit sign or zero extended representation of the 32-bit value.

3.7.4 Explicit cast improperly applied

Expressions can sometimes be very tricky because of the conversion rules. Therefore, you should be very explicit about how you want the expression to be evaluated by adding casts wherever necessary. Consider the following example:

```
01  int main(int argc, char *argv[])
02  {
03    int i, val;
04    long l;
05
06    val =        l / i;   /* implicit truncation to 32 bits */
07    val = (int)  l / i;   /* l is truncated to 32 bits before division */
08   val = (int) (l / i);  /* result of division is truncated to 32 bits */
09  }
```

The code example above shows the division of a long integer variable l by an integer variable i. The result is then assigned to the integer variable val. The effective evaluation of the expressions in lines 6 through 8 is:

- In line 6, the expression l / i is evaluated with data type long and the result is then implicitly truncated to int before assignment to val.
- In line 7, the expression l / i is evaluated with data type int due to the explicit cast to operand l, and the result is then assigned to val.
- In line 8, the expression l / i is evaluated with data type long and the result is then explicitly truncated to int before assignment to val.

3.7.4.1 Recommendation

Explicit narrowing casts should be used on expressions, not operands, so that the expression may be evaluated with greater precision if any of the operands involved has greater precision than the explicit cast, therefore forcing all other operands to be converted to its data type.

Line 8 illustrates a properly applied explicit cast that will yield identical results to line 6.

Line 7 shows a similar statement with an explicit cast, but in this case the result of the expression may be different, as the cast is applied to the long variable before the division.

3.7.4.2 Compiler and lint assistance

Both the VisualAge C compiler and `lint` will help you spot occurrences where you might want to change the evaluation of expressions by applying explicit casts. The relevant compiler output (with the -q64 -qwarn64 switches) is:

```
line 6.18: 64-bit portability: possible loss of digits through conversion
of long int type into int type.
line 7.9: 64-bit portability: possible loss of digits through conversion of
long int type into int type.
line 8.9: 64-bit portability: possible loss of digits through conversion of
long int type into int type.
```

Here is the relevant `lint` output:

```
line 6: warning: conversion from long may lose accuracy
line 7: warning: conversion from long may lose accuracy
line 8: warning: conversion from long may lose accuracy
```

3.8 Pointer assignment and arithmetic

When migrating from 32-bit environments to 64-bit environments, it is crucial to avoid pointer corruption. Some of the possible problems are:

- Assigning an int (32 bits) or a 32-bit hexadecimal constant to a pointer type variable (64 bits) or casting a pointer to an int will yield an invalid address, and will cause errors when the pointer is dereferenced. Also, the comparison of an int to a pointer may cause unexpected results.
- Pointers are converted to int or unsigned int with the expectation that the pointer value will be preserved, as casting a pointer to an int will result in data truncation.
- Without proper function prototypes, functions that return pointers will return truncated return values, as the functions are implicitly declared to return an int that is just 32 bits, instead of the expected 64 bits of a pointer.

- The code assumes that pointers and int are the same size in an arithmetic context, as pointer arithmetic usually is a source of problems in migration. The ANSI C standard dictates that incrementing a pointer yields adding the size of the data type to which it points to the pointer value. For example, if the variable p is a pointer to long, then the operation (p+1) increments the value of p by 4 bytes in ILP32 mode and by 8 bytes in LP64 mode. Therefore, casts between long* to int* are problematic because of the size differences of pointer objects (32 bits versus 64 bits).

3.8.1 Different byte sizes for int and pointers in LP64 mode

The assumption that an int and a pointer have the same size is not true for the LP64 mode. Because ints and pointers are the same size in the ILP32 environment, a lot of code falsely relies on this assumption. Pointers are often cast to int or unsigned int for address arithmetic. Instead, pointers can be cast to long because long and pointers are the same size in both ILP32 and LP64 worlds. Rather than explicitly using unsigned long, you should use the predefined data type uintptr_t (from <inttypes.h>) instead, because it makes the code more portable and therefore safe against future changes. Consider the following code example:

```
01  #define PAGESIZE 4096
02
03  int main(int argc, char *argv[])
04  {
05    unsigned char *p;
06
07    p = (unsigned char *) ((unsigned int) p + PAGESIZE);
08  }
```

3.8.1.1 Compiler assistance

The Visual Age C compiler with the -qwarn64 switch turned on helps you identify such errors. These are the compiler warnings for line 7:

```
64-bit portability:  possible truncation of pointer through conversion of
                     pointer type into unsigned int type
64-bit portability:  possible incorrect pointer through conversion of
                     unsigned int type into pointer
```

3.8.1.2 Recommendation

The solution to the above problem is to change the program to use the generic pointer type uintptr_t (line 9) from <inttypes.h> (line 1):

```
01  #include <inttypes.h>
02
03  #define PAGESIZE 4096
```

```

int main(int argc, char *argv[])
{
  unsigned char *p;

  p = (unsigned char *) ((uintptr_t) p + PAGESIZE);
}
```

3.8.2 Assignment of 64-bit pointer value to a smaller integral type

As in the case of an LP64 long, assignment of a LP64 pointer value to a 32-bit data type variable will result in truncation of the pointer value. The pointer value cannot be reconstructed from the int or unsigned int. If the address value being converted is in the range of [0..2^{32}-1], which is the unsigned range of 32 bits, the code may appear to work only to fail whenever an address is used beyond the 4 GB memory range.

Since ANSI-conforming C compilers are required to provide a diagnostic, usually a warning, for integral to pointer and pointer to integral assignments, existing source code is likely to have an explicit cast on these assignments. These explicit casts have been introduced to suppress the diagnostics from the compiler and `lint`. In LP64 mode, if the conversions are to any type less than 64 bits, these conversions are very likely to be a source of porting problems. The example in Figure 14 on page 51 shows a combination of explicit and implicit pointer to integer conversions that could ultimately lead to problems.

```
extern void dosomething_int(int, int);
extern void dosomething_long(long, long);

int main(int argc, char *argv[])
{
  int i;
  long l;
  int *ptrint;
  void *ptrvoid;

  i = ptrint;         /* implicit loss of upper 32 bits */
  i = ptrvoid;        /* implicit loss of upper 32 bits */
  i = (int) ptrint;   /* explicit loss of upper 32 bits */
  i = (int) ptrvoid;  /* explicit loss of upper 32 bits */

  dosomething_int(ptrint, (int) ptrvoid);

  l = ptrint;
  l = ptrvoid;
  l = (int) ptrint;   /* implicit loss of upper 32 bits */
  l = (int) ptrvoid;  /* implicit loss of upper 32 bits */

  dosomething_long(ptrint, (int) ptrvoid);
}
```

Figure 14. Truncation of a 64-bit pointer value

3.8.2.1 Recommendation

Code involving conversions of pointers from or to integral values should be reviewed. If these pointer to integral conversions are absolutely necessary, the integral type should be either long or unsigned long and an explicit cast to long or unsigned long should be used. Even better would be to use the generic pointer data type uintptr_t from <inttypes.h>.

Fully prototyped function declarations should be in scope at the point of all calls, allowing the C compiler and `lint` to scrutinize pointer to integral conversions of function arguments and return values. See Section 3.11, “Lack of function prototypes” on page 75 for more information.

3.8.2.2 lint assistance

The lint code checking tool will not only flag all occurrences of non-conforming implicit and explicit pointer to integer conversions, but also flag all explicit conversions that may lose significant bits. Here is the relevant output of examining the code shown in Figure 14 with `lint`:

```
line 11: warning: illegal combination of pointer and integer, op =
line 11: warning: conversion from "PTR int" may lose accuracy
line 12: warning: illegal combination of pointer and integer, op =
line 12: warning: conversion from "PTR void" may lose accuracy
line 13: warning: conversion from "PTR int" may lose accuracy
line 14: warning: conversion from "PTR void" may lose accuracy
line 16: warning: mismatched type in function argument
line 16: warning: illegal combination of pointer and integer, op PARAMETER
line 16: warning: conversion from "PTR void" may lose accuracy
line 16: warning: conversion from "PTR int" may lose accuracy
line 18: warning: illegal combination of pointer and integer, op =
line 19: warning: illegal combination of pointer and integer, op =
line 20: warning: conversion from "PTR int" may lose accuracy
line 21: warning: conversion from "PTR void" may lose accuracy
line 23: warning: mismatched type in function argument
line 23: warning: illegal combination of pointer and integer, op PARAMETER
line 23: warning: conversion from "PTR void" may lose accuracy
```

Implicit pointer to integer conversion takes places in lines 11, 12, and 16, while explicit pointer to integer conversion occurs in lines 13, 14, 16 (second argument), 20, 21, and 23 (second argument).

3.8.3 Assumption about pointers and int in arithmetic context

Pointers may not be used directly with arithmetic or bitwise operators, with the exception of adding and subtracting integer values, and computing the difference between two pointers. There are times when a pointer must be explicitly cast to an integral type to be used with these operators. Such an example would be the AIX 5L kernel's use of bitwise shifts and bitwise AND operations to determine the memory segment containing a particular address. These explicit casts should be to either long or unsigned long, which will preserve the 64-bit values in LP64 mode and the 32-bit values in ILP32 mode. Consider the example in Figure 15 on page 53, which tries to mimic the scenario mentioned previously.

```
01  #include <inttypes.h>
02
03  #define BUSY 01
04
05  struct blk_t { ..... } blk;
06
07  void dosomething(void)
08  {
09    struct blk *p, *word;
10
11    word = (struct blk *) (((int) p) | BUSY);
12
13    word = (struct blk *) (((unsigned long) p) | BUSY);
14
15    word = (struct blk *) (((uintptr_t) p) | BUSY);
16  }
```

Figure 15. Wrong assumption about pointer and integer size

3.8.3.1 Recommendation

The code on line 11 assumes that the size of a pointer is the same as an int, which is true in ILP32 mode but false in LP64 mode. The result is that the 64-bit pointer p is converted to a 32-bit value and ORed with the BUSY bit. The 32-bit integer result is then cast back to the pointer p, so the upper 32 bits of its address have been lost. The correct way to perform this operation is:

- To cast the pointer to an unsigned long (as pointers and long integers (long) are guaranteed to have the same byte size as each other in both ILP32 and LP64 mode), as shown in line 13.

  or

- To cast the pointer to the generic pointer data type uintptr_t as defined in the ANSI C header file <inttypes.h>. This data type is guaranteed to have the same byte size as a pointer in both ILP32 and LP64 modes, as demonstrated in line 15.

3.8.3.2 lint assistance

`lint` will help you to find occurrences of this type of error, as it will flag pointer to integer conversions, which may result in loss of bits. The relevant `lint` output is:

```
line 11: warning: conversion from "PTR strty(036)" may lose accuracy
line 13: warning: bitwise " OR " involving a "ulong"
line 15: warning: bitwise " OR " involving a "ulong"
```

The last two lines of the `lint` output do not indicate a possible error; they just warn you that you are about to perform bit manipulation with a pointer value (lines 13 and 15).

3.8.4 Address arithmetic and pointer arithmetic

In general, using pointer arithmetic is preferable to using address arithmetic, because pointer arithmetic is independent of the data model, whereas address arithmetic may not be. Pointer arithmetic usually leads to simpler code. However, if the assumption that an int and a pointer have the same size has been made (which is not true for LP64 mode), then address arithmetic may fail when pointer arithmetic is independent of the employed data model. Consider the following code example:

```
01 #define ADD_NUM_PTRS 100
02
03 int *start;
04 int *end;
05
06 start = (int *) malloc(4 * ADD_NUM_PTRS);
07 end = (int *) ((unsigned int) start + 4 * ADD_NUM_PTRS);
```

3.8.4.1 Recommendation

Instead of using address arithmetic, it is better to use pointer arithmetic, as it is:

- Independent of the employed data model
- Easier to read and understand

Therefore, the above code example should be changed to:

```
01 #define ADD_NUM_PTRS 100
02
03 int *start;
04 int *end;
05
06 start = (int *) malloc(sizeof(start) * ADD_NUM_PTRS);
07 end = p + ADD_NUM_PTRS;
```

Not only is the incorrect assumption that a pointer to an int occupies only 4 bytes (as done in lines 6 and 7) corrected, but the truncation of the new value for the pointer end during the casting to unsigned int is corrected as well.

3.8.4.2 lint assistance

`lint` will help you to find occurrences of this type of error, as it will flag pointer to integer conversions that may result in loss of bits. The relevant `lint` output is:

```
line 7: warning: conversion from "PTR int" may lose accuracy
```

3.8.5 Pointer to int is incompatible with pointer to long

Pointers to different data types are not compatible in C, and a pointer to one type should not be assigned to a pointer of another type. For historical reasons, however, most compilers do not stringently enforce this restriction and comply with the ANSI/ISO C Standard by issuing a warning. In source code, where int and long have been used interchangeably, pointers to int and long may have also been used interchangeably. In LP64 mode, these point to objects of different size, and subsequent dereference of such a pointer will clearly result in undefined behavior. Consider the code example shown in Figure 16.

```
01  extern void dosomething_int(int *);
02  extern void dosomething_long(long *);
03
04  int main(int argc, char *argv[])
05  {
06    int *iptr1, *iptr2;
07    long *lptr1, *lptr2;
08
09    lptr1 = iptr1;
10    iptr2 = lptr2;
11
12    lptr1 = (long *) iptr1;
13    iptr2 = (int *) lptr2;
14
15    dosomething_int(lptr1);
16    dosomething_long(iptr1);
17  }
```

Figure 16. Pointers to different data types are not compatible

Both the C compiler and `lint` report the incompatibility of pointer assignments in lines 9, 10, 15, and 16 in Figure 16. In addition to the obvious object size mismatches that would occur if these pointers were dereferenced, the alignment requirements for int and long are different in LP64 mode. If a pointer to long is used to reference a memory address that is not 8-byte aligned, an alignment fault will occur. This will, in turn, require intervention by the operating system and degrade application performance.

Lines 12 and 13 in Figure 16 on page 55, while valid C code, still present the same problems as lines 9 and 10. The explicit casts have probably been introduced into the source code to suppress the warnings from the C compiler and `lint`.

Any pointer type may be assigned to or from a void*. Any code which effectively assigns an int* to a long*, or the reverse, through an intermediate void* variable or function parameter may exhibit the undefined behavior possible in the above example.

3.8.5.1 Recommendation

Examine all instances of incompatible pointer assignments, particularly those involving a long* data type. The type of the object pointed to should be made consistent, and that choice should be based on the range of values to be held by the object.

For cases where the types pointed to are intentionally different, as with char* pointers returned from older memory allocation or memory management routines, use an explicit cast to indicate to the compiler and to `lint` that this is intentional. A better solution is to bring the code up to ANSI C specifications and use void* for generic pointers. Remember to take any alignment issues into account.

3.8.5.2 Compiler and lint assistance

Both the C compiler and lint will help you identify incompatible pointer assignments. The compiler in ANSI mode (`xlc` or `c89`) actually will refuse to compile the example Figure 16 on page 55 because of these incompatible pointer assignments (which are considered errors). Here is the relevant compiler output for `xlc -q64 -qwarn64 filename.c`:

```
line 9.9: Operation between types "long*" and "int*" is not allowed.
line 10.9: Operation between types "int*" and "long*" is not allowed.
line 15.19: Function argument assignment between types "int*" and "long*"
is not allowed.
line 16.20: Function argument assignment between types "long*" and "int*"
is not allowed.
```

However, the compiler in extended mode (`cc`) will compile the code example above and will only issue warnings instead of errors.

Here is the relevant `lint` output:

```
line 9: warning: illegal pointer combination, op =
line 10: warning: illegal pointer combination, op =
line 15: warning: illegal pointer combination, op PARAMETER
line 16: warning: illegal pointer combination, op PARAMETER
```

3.9 Integer constants

A loss of data can occur in some constant expressions because of lack of precision. These types of problems are very hard to find and may have gone unnoticed so far. You should therefore be very explicit about specifying the type(s) in your constant expressions and add some combination of qualifiers {u,U,l,L} to the end of each integer constant to specify exactly its type. You might also use casts to specify the type of a constant expression.

3.9.1 ANSI C rules for integer constants

Integer constants may be specified in decimal, octal, or hexadecimal notation. Figure 17 on page 58 shows the ANSI C language syntax definition for integer constants. The rules for determining the radix of an integer constant are:

- If the integer constant begins with the letters 0x or 0X, then it is in hexadecimal notation, with the characters a through f (or A through F) representing the numbers 10 through 15.
- If the integer constant begins with the digit 0, then it is in octal notation. Leading and high-order zeroes only serve to denote octal notation and have no other effect.
- Otherwise, it is in decimal notation.

The letters l or L may immediately follow the integer constant to indicate a constant of type long. The lower case letter l should not be used as it can be easily confused with the digit 1. The suffix letters u or U indicate an unsigned constant.

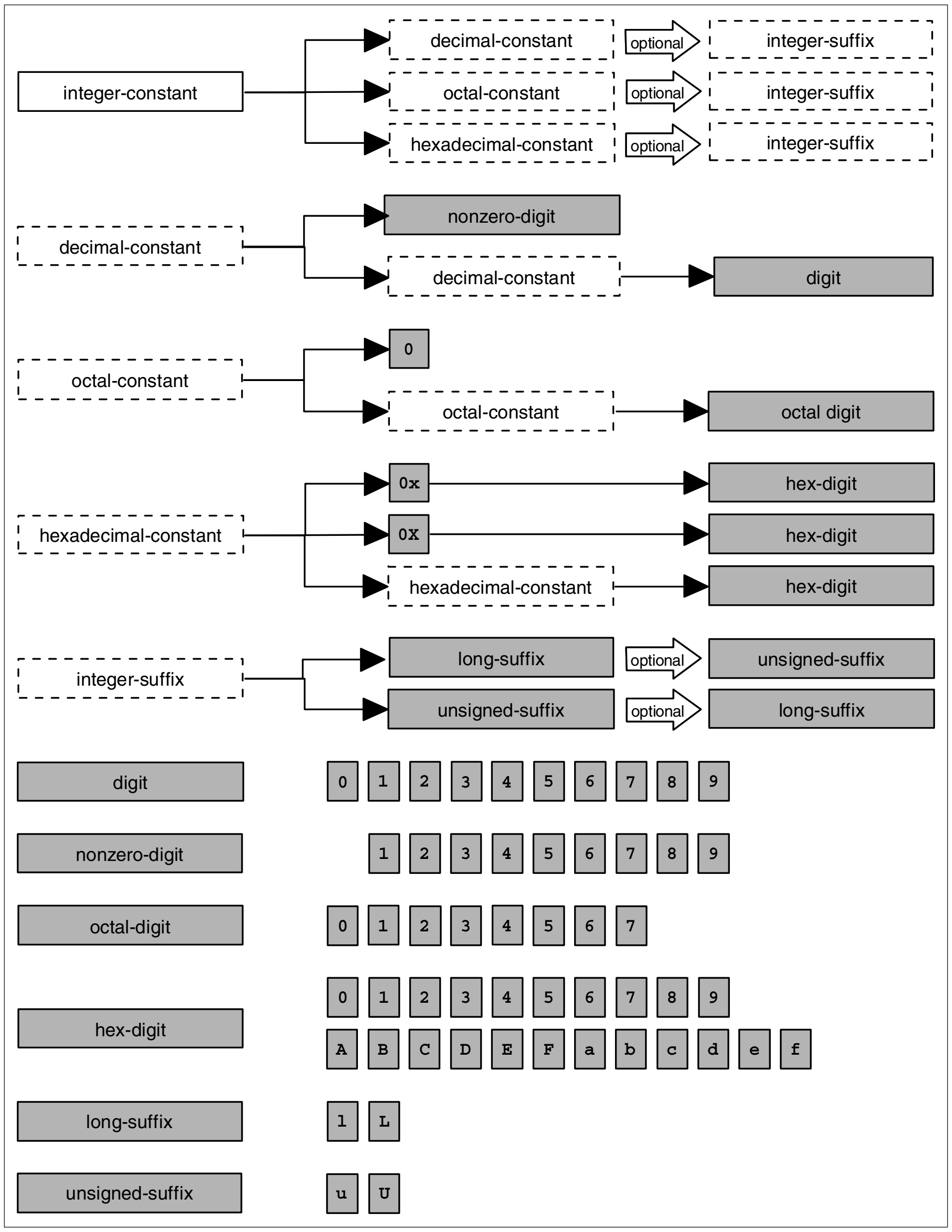


Figure 17. ANSI C language syntax definition for integer constants

Porting code that uses integer constants must:

- Consider that integer constants may be more than 32 bits.

- Do not assume that long or unsigned long data is 32 bits.
- Do not depend on the specific behavior at an assumed data type length.

Integer constants can have different values on 32-bit and 64-bit systems. Also, when a program with hexadecimal constants is ported from ILP32 mode to LP64 mode, the data types assigned to the constants may change.

3.9.2 Untyped integral constants are int by default

The ANSI C standard states that the type of an integer constant, depending on its format and suffix, is the first (read: smallest) type in the corresponding list that will hold the value. The quantity of leading zeros does *not* influence the type selection. See Table 10 for types of integer constants and their assigned ANSI C data type.

Table 10. Types of integer constants and their assigned ANSI C data type

Suffix	Data type
Unsuffixed decimal number	int, long, unsigned long
Unsuffixed octal or hexadecimal number	int, unsigned int, long, unsigned long
Suffixed by u or U	unsigned int, unsigned long
Suffixed by l or L	long, unsigned long
Suffixed by both u or U and l or L	unsigned long

Code may behave differently when compiled for LP64 mode than when compiled for ILP32 if it:

- Does not take into consideration that integral constants may be represented as 32-bit types, even when used in expressions with 64-bit types.
- Assumes that long or unsigned long data is 32 bits in length.
- Depends on specific behavior at an assumed data type length.

Table 11 lists some common integer constants and their assigned data types for ILP32 mode and LP64 mode.

Table 11. Common integer constants and their types in ILP32 and LP64

Constant	Value	ANSI C ILP32	ANSI C LP64
0x7FFFFFFF	2^{31}-1	int	int
0x7FFFFFFFL	2^{31}-1	long	long

Constant	Value	ANSI C ILP32	ANSI C LP64
0x80000000	2^{31}	unsigned int	unsigned int
0x80000000L	2^{31}	unsigned long	long
0xFFFFFFFF	2^{32}-1	unsigned int	unsigned int
0xFFFFFFFFL	2^{32}-1	unsigned long	long

Table 12 lists some common integer constants and their different values for ILP32 mode and LP64 mode.

Table 12. Common integer constants and their values in ILP32 and LP64

Constant	Value	ANSI C ILP32	ANSI C LP64
0x7FFFFFFF	2^{31}-1	2,147,483,647	2,147,483,647
0x7FFFFFFFL	2^{31}-1	2,147,483,647	2,147,483,647
0x80000000	2^{31}	2,147,483,648	2,147,483,648
0x80000000L	2^{31}	2,147,483,648	2,147,483,648
0xFFFFFFFF	2^{32}-1	4,294,967,295	4,294,967,295
0xFFFFFFFFL	2^{32}-1	-1	4,294,967,295
4294967296	2^{32}	0	4,294,967,296
0x100000000	2^{32}	0	4,294,967,296
0xFFFFFFFFFFFFFFFF	2^{64}-1	-1	-1

3.9.3 General guidelines

The usage of all constants, including symbolic constants established with preprocessor `#define` statements, should be reviewed. Special attention should be given to:

- long or unsigned long expressions containing constants used in integer subexpressions which may overflow the maximum or underflow the minimum values expressible in 32 bits
- Expressions containing octal or hexadecimal constants whose high order bit is 2^{31}
- Expressions depending on truncation at bit 32 on an overflow
- Left shift expressions that assume truncation at bit 32

The following sections illustrate the cases above.

3.9.4 Integer expression with overflow in 64-bit expression

Consider the following code example:

```
01  long l1, l2;
02
03  l2 = l1 + 20000000 * 30000000;         /* 32-bit multiplication */
04
05  l2 = l1 + 20000000L * 30000000;        /* 32-bit multiplication */
06  l2 = l1 + (long) 20000000 * 30000000;  /* 64-bit multiplication */
07  l2 = l1 + 20000000 * (long) 30000000;  /* 64-bit multiplication */
```

This is a special form of an integer expression with overflow being used in a 64-bit expression. The two constants in line 3 are 32-bit integer constants, and the integer multiplication results in a 32-bit overflow with the truncated folded constant value of 1,658,683,392 added to the variable l2.

3.9.4.1 Recommendation

By using the type suffix L to specify a long type on at least one of the constants, as shown in line 5, the multiplication will be done with 64-bit constants. Similarly, either constant could have been explicitly cast to a type of long (as shown in lines 6 and 7).

3.9.4.2 Compiler and lint assistance

Unfortunately, neither the VisualAge C compiler or `lint` will give you any warning about the integer expression overflow when compiling for LP64 mode. The C compiler, however, when compiling for ILP32 mode, will give you an error message saying that the expression `20000000 * 30000000` does not evaluate to a constant that fits in its signed type.

3.9.5 Hexadecimal constants

Consider the following code example:

```
01  long l;
02
03  l &= ~(0x80000000);          /* turns off left most 33 bits */
04  l &= ~(0x0000000080000000);  /* turns off left most 33 bits */
05
06  l &= ~((long) 0x80000000);   /* turns off bit 2**31 */
07  l &= ~(0x80000000L);         /* turns off bit 2**31 */
```

This piece of code illustrates a hexadecimal constant with the 2^{31}-bit set. Because the significant bits of the constant in line 3 will fit into 32 bits, it has a type of unsigned int. The bitwise complement will yield an unsigned int constant of value 0x7fffffff. In ILP32 mode, the assignment and operator would effectively turn off the 231-bit. In LP64 mode, the unsigned int would be converted to a long with the value 0x000000007fffffff, effectively turning off 33 bits of the value of the variable l. Line 4 would have the same result, since leading zeros are insignificant in determining the data type of the constant.

3.9.5.1 Recommendation

Lines 6 and 7 of the previous example show the use of either an explicit cast or a type suffix, respectively, to be certain that the constant is treated as a 64-bit value. These two lines will turn off the 2^{31}-bit in both LP64 and ILP32 mode.

3.9.5.2 Compiler and lint assistance

Both the Visual Age C compiler and `lint` can help you detect these kind of errors. Here are the relevant compiler warnings (when compiled with -q64 and -qwarn64):

```
line 3.7: 64-bit portability: possible change of result through conversion
of unsigned int type into long int type.
line 4.8: 64-bit portability: constant which selected unsigned long int in
32-bit mode may select long int in 64-bit mode
```

Here is the relevant output of `lint` (when used with the -t switch), which just informs you about the bitwise AND manipulation of a long variable:

```
line 3: warning: bitwise " AND " involving a "long"
line 4: warning: bitwise " AND " involving a "long"
line 6: warning: bitwise " AND " involving a "long"
line 7: warning: bitwise " AND " involving a "long"
```

3.9.6 Code depending on truncation at 32 bits on overflow

Consider the following code example:

```
01  long l;
02
03  l += 0xffffffff;
```

The constant is a 32-bit unsigned int with a value of 4,294,967,295 in both ILP32 and LP64 mode. The addition is done as an unsigned long, which is cast to a type long. In ILP32 mode, the result has a value of l – 1 because of the truncation to 32 bits (always assuming a twos-complement system for negative integers). In LP64 mode, the addition result is a 64-bit long with a value of l + 4,294,967,295, which is certainly not the expected result.

3.9.6.1 Recommendation

You should rewrite your code and use explicit casts to make sure you obtain the desired result. To obtain the twos-complement independently from the data model, the previous code example should be rewritten as:

```
01  long l;
02
03  l += ((1L << (8 * sizeof(long))) - 1);
```

Here the number 8 equals the number of bits per char and should be represented as an architecture dependent `#define` (CHAR_BIT in <limits.h>).

3.9.6.2 Compiler assistance

The VisualAge C compiler can help you detect these kind of errors when compiling with -q64 and -qwarn64. Here is the relevant compiler warning:

```
line 3.6: 64-bit portability: possible change of result through conversion
of unsigned int type into long int type.
```

3.9.7 Wrong assumption about size of long integers

Consider the following code example:

```
01  long l1, l2;
02
03  l2 = (l1 << 5) >> 16;  /* depends on truncation at bit 32 */
04
05  l2 = (l1 << (8 * sizeof(l1) - 27)) >> (8 * sizeof(l1) - 16);
```

This is an example of code that presumes to know the number of bits in a long data type. The code is attempting to extract bits 11-26 from the long variable l1 as a signed quantity. The left shift in line 3 is dependent on truncation occurring at bit 32, and while the code will work in ILP32 mode, it will not port to LP64 mode. Code which assumes to know the size of any data type other than char is not portable. Code that assumes the size of a long or unsigned long data type will certainly be a problem when ported to LP64 mode.

3.9.7.1 Recommendation

The code in line 5 will yield identical results in both LP64 and ILP32 modes, where:

- 8 = number of bits per char and should be represented as an architecture dependent `#define` (CHAR_BIT in <limits.h>)
- 27 = one more than the highest bit desired in the result
- 16 = size of the field being extracted

3.9.7.2 lint assistance
Unfortunately, the VisualAge C compiler will not give you any warnings regarding the previous code as it cannot know your wrong assumption. Also, `lint` with the `-t` option will just give you a general warning about left/right shift operation involving a long variable:

```
line 3: warning: Left Shift involving a "long"
line 3: warning: Right Shift involving a "long"
line 5: warning: Left Shift involving a "long"
line 5: warning: Right Shift involving a "long"
```

3.9.8 Bit shifts and bit masks
Bit shifts and bit masks are sometimes coded with the assumption that the operations are performed in variables that have the same data type as the result. In cases such as”

```
z = x operation y
```

the data type used for the intermediate result of the operation depends on the data types of y and x. The data type of the intermediate result is then converted according to the ANSI C integer conversion rules (see Section 3.5, “ANSI C integer conversion rules” on page 31) to the data type of z. If the result of the operation requires 64 bits, but the data types of x and y are only 32-bit, then the intermediate result will either overflow or be truncated before being assigned to z. Consider the following code example:

```
01  int i = 32;
02  long j;
03
04  j = 1 << i;  /* j will be 0 because RHS is integer expression */
05
06  j = 1L << i;
07  j = (long) 1 << i;
```

The left operand 1 is an integer constant that the compiler treats as a 32-bit value in both ILP32 and LP64 mode. The bit shift in line 4 uses a 32-bit data type (int) for the intermediate result (as both operands are 32-bit data types). Therefore, in both data models, the operation overflows. With the truncation at bit 32, the final result of the left shift operation is 0. Note that only the left operand of a shift operator determines the data type of the result. The data type of the shift count operand is irrelevant.

3.9.8.1 Recommendation
By using the type suffix L (or UL) to specify a long (or unsigned long) type on the constant 1, as shown in line 6, the left shift will be done with 64-bit

constants. Similarly, the constant could have been explicitly cast to a type of long (as shown in line 7). In both cases, the expected result of 0x100000000 is obtained.

3.9.8.2 Compiler and lint assistance

Unfortunately, neither the VisualAge C compiler or `lint` will give you any warning about the integer expression overflow when compiling for LP64 mode.

3.10 C and C++ data type alignment issues

This section describes the C language data types of AIX 5L on both Power and Itanium-based systems. It also describes the differences in *alignment* between the data types in 32-bit and 64-bit programming models, along with the porting issues that may be encountered, and methods that can be used to write programs so that they are not impacted by those differences.

3.10.1 C and C++ data type alignment in AIX 5L

Support for a 64-bit address space and larger scalar arithmetic ranges in LP64 mode naturally requires changes in at least some of the basic C and C++ data types. Details of the alignment characteristics of C and C++ language base data types in each programming model and hardware platform are shown in Table 13.

Table 13. Data type alignment in bytes for AIX 5L

	AIX 5L for Power		**AIX 5L for Itanium-based systems**	
Data type	**ILP32 model**	**LP64 model**	**ILP32 model**	**LP64 model**
char	1	1	1	1
short	2	2	2	2
int	4	4	4	4
long	4	8	4	8
long long	8	8	4	8
float	4	4	4	4
double	4	4	4	8
long double	8	8	4	16
pointer	4	8	4	8

The differences between the LP64 programming model and the others is the alignment of long, long long, pointer, and long double data types. As shown in Table 13 on page 65, pointers, long and long long integers in LP64 mode are aligned at 8 bytes (64 bits) boundaries. The alignment restrictions for these data types, as well as the size of long double floating point types, have changed for performance considerations.

Although the Power model supports 128-bit long double variables with 128-bit alignment, when the -qlongdouble option is used with the compiler, the default alignment for long double is 8 bytes (64 bits).

3.10.2 Data alignment

Most processors require every data item in memory to be aligned on 2-, 4-, or 8-byte boundaries; otherwise, they may suffer performance degradation by having to perform multiple read operations, or they may have to raise a hardware exception so that the operating system will handle it, with additional performance degradation. To solve this misalignment problem, the compiler adds filler bytes called *padding* immediately before every misaligned item to ensure it is aligned properly on the correct boundary. Although the padding bytes added by the compiler are invisible to the application code, they do exist and can cause the layout of a data structure in memory to differ from what is expected.

For example, consider the structure definition shown below:

```
struct {
   char     cmdcode;     /* 1 byte command code */
   double   param;       /* 8 bytes parameter */
   short    retcode;     /* 2 bytes return code */
} cmdstruct_t;
```

Figure 18 on page 67 shows the layout in memory for the ILP32 and LP64 data model.

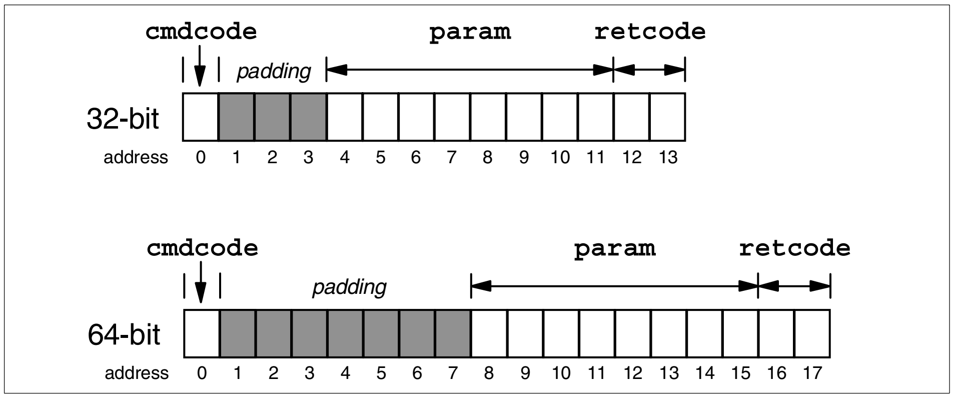


Figure 18. Different structure padding in ILP32 and LP64 mode

Each field in the structure has the same size in both 32-bit and 64-bit environments, indicating that there should not be a problem. However, if we examine the overall size and layout of the structure in 32-bit and 64-bit environments, we can see that they do in fact differ. This is due to different padding being used in each environment to achieve the correct alignments for the data types in the structure. If the structure is shared or exchanged among 32-bit and 64-bit processes, the data fields and padding of one environment will not match the expectations of the other.

The following sections deal with a number of techniques that can be used to solve this problem.

3.10.3 Data reordering

You can reorder the fields in the data structure to get the alignments in both 32-bit and 64-bit environments to match. Using the example shown in Section 3.10.2, “Data alignment” on page 66, if the structure members are reordered, the resulting structure is the same size in both 32-bit and 64-bit environments. The structure would have to be reordered, as in the following code example:

```
struct {
   double   param;      /* 8 bytes parameter */
   short    retcode;    /* 2 bytes return code */
   char     cmdcode;    /* 1 byte command code */
} cmdstruct_t;
```

Figure 19 on page 68 shows the rearranged structure layout in memory.

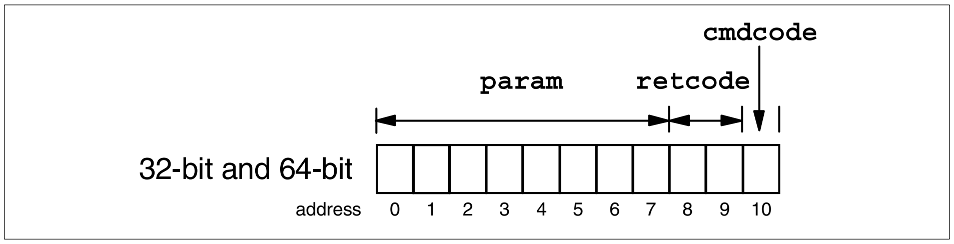


Figure 19. Rearranged structure to match the alignment in ILP32 and LP64 mode

The success of this method will depend on the data types used in the structure and the way in which the structure as a whole is used. For example, if the structure is defined in a header file by another device driver or kernel component for which you do not have the source code, you will not be able to reorder the structure members.

Internal data structures in applications should always be checked for holes. A simple rule for reordering the structure is to move the long and pointer fields to the beginning of the structure, as they will grow to 64 bits in the LP64 model.

However, a structure in which every member is a data type that has a different alignment or size in 32-bit and 64-bit environments will not benefit from data reordering alone.

3.10.4 User-defined padding

If you are unable to reorder the members of a structure, or if reordering alone cannot provide correct alignment, another method that can be used is to introduce user-defined padding. The user-defined padding technique can be used in conjunction with data reordering (if required). Depending on the data types involved, a conditional compile section may be necessary.
A conditional compile section will be required when the structure uses data types that have different sizes in the 32-bit and 64-bit environments. Using the structure shown in Section 3.10.2, “Data alignment” on page 66 as an example, rather than reordering the members, an additional member could be added, which would cause both 32-bit and 64-bit environment versions of the structure to have identical alignment. In this case, an additional structure member of type int placed between the first and second structure members would achieve the objective shown in the code example below:

```
struct {
   char     cmdcode;    /* 1 byte command code */
   int      usrpad;     /* 4 bytes user defined padding */
   double   param;      /* 8 bytes parameter */
   short    retcode;    /* 2 bytes return code */
} cmdstruct_t;
```

Figure 20 shows the memory layout of the structure with user-defined padding.

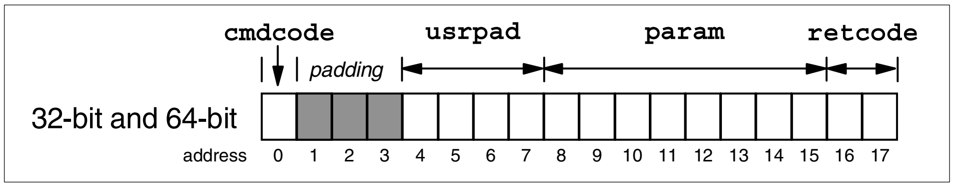


Figure 20. User-defined structure padding

Unlike padding added by the compiler, the user-defined padding is visible in the program address space and thus requires a unique symbol name within the structure. Depending on the sizes and alignments of the adjacent structure members, user-defined padding of char, short, and int can be used.

If the data structure contains variables of type long or pointer, which have different sizes in the 32-bit and 64-bit environments, a conditional compile section will be required. For example:

```
struct {
#ifdef __64BIT__
   long  *ptr;     /* 8 bytes in LP64 */
#else
   long  *ptr;     /* 4 bytes in ILP32 */
   long  padding;  /* 4 bytes in ILP32 */
#endif
   int   count;
} pointer_t;
```

In this example, the overall size of the structure will be the same in both environments, as will the alignments of the ptr and count variables.

Any code that uses a structure defined in this way must be aware of the process environment in which it is running (32-bit or 64-bit) and the environment of the source of the structure, so that it can correctly interpret the data fields that have a different size in each environment.

3.10.5 Determining structure alignment

You can verify the layout of a data structure with a handy macro, offsetof(type,member), which is defined in the standard header file

<stddef.h>. The macro expands to an integral constant expression (of type size_t) that is the offset (in bytes) of the specified member within the data structure. You can use the macro in a small stub program, that you compile and run in both 32-bit and 64-bit environments, to determine that the structure offsets are the same. For example:

```
#include <stddef.h>

struct mystruct_t {
  char filler;
  int suspect;
};

int main(int argc, char *argv[])
{
  printf("offset is %ld\n", offsetof(struct mystruct_t, suspect));
}
```

If offsetof() is not defined in a non-ANSI implementation (for example, your origin platform), it is possible to define it as follows:

```
#define offsetof(type,member) ((size_t)&((type *)0)->member)
```

If the implementation does not permit the use of the null pointer constant in this fashion, it is possible to compute the offset by using a predefined, non-null pointer and subtracting the member's address from the structure's base address. For example:

```
struct mystruct_t {
  char filler;
  int suspect;
};

int main(int argc, char *argv[])
{
  struct mystruct_t x;

  printf("offset = %ld\n",
         (unsigned long) &x.suspect - (unsigned long) &x);
}
```

3.10.6 Objects change size

Data objects that contain pointer, long, long long, or long double data types will have different sizes in ILP32 and LP64 modes. This change in data structure sizes may not be a problem. If the data is to be solely used by the binary that produced the data or another program compiled for the same compilation model, the size difference is not an issue, with the exception of a potential size problem. A problem does exist where an LP64 model binary must consume data produced by a ILP32 model binary or where the data flow is in the opposite direction.

For structures that must be passed between 32-bit and 64-bit environments, there are two approaches that can be taken:

- Alter the structure using the methods described previously, so that the structure has identical size and alignment in both environments.
- Leave the structure definition as is, which results in differences between the 32-bit and 64-bit version. Each segment of code that handles the structure must determine the type of environment that created the structure and take appropriate action.

If the structure is private to your application, either approach may be taken. If the structure is defined elsewhere, for example, as part of a POSIX or XPG standard, you may be unable to alter the structure definition to ensure that the sizes and alignments match in both 32- and 64-bit environments.

With careful design, compatible data structures can be defined to allow sharing of data between binaries from different models. The preprocessor directive __64BIT__ will allow the declaration of a structure that is binary data compatible between compilation models:

```
struct mysharedstruct_t {
#ifdef __64BIT__
   long        64bit_value;
   int         32bit_value;
   int         other_32bit_value;
   long double big_fp_value;
#else    /* !__64BIT__ */
   long long   64bit_value;
   long        32bit_value;
   int         other_32bit_value;
   long double big_fp_value;
   int         padding;
#endif   /* __64BIT__ */
};
```

The above structure illustrates a C data structure that is binary data compatible in either ILP32 or LP64 mode. The declaration preserves the alignment and size of each structure member.

Sharing of pointer values between ILP32 and LP64 applications is meaningless. In serious database-oriented applications, pointers rarely appear in declarations of data written to mass storage devices. These applications are normally concerned about the efficient use of storage, and already avoid pointers. File offsets can be expressed in terms of the 64-bit data type available in each model.

In cases where an LP64 program must deal with an ILP32 data structure that contains pointers, more effort is required. Assuming that data written out in pointer fields is irrelevant or expressed in terms of some offset, and will be filled in when the structure is memory resident, a new data structure that encapsulates the old data can be defined. Care must be taken to preserve the alignments in the old structure.

3.10.7 __align specifier

On the Power version on AIX 5L, you have the option of using the __align specifier, which can be used to explicitly specify alignment and padding when declaring or defining data items. The syntax for the __align specifier can be seen below:

```
declarator __align (integer_constant) identifier;
struct_or_union_specifier __align (integer_constant) [identifier
{struct_declaration_list}];
```

where:

integer_constant specifies a byte-alignment boundary, which is an integer constant, greater than 0 and which is a power of 2.

The __align specifier can only be used with declarations of first-level variables and aggregate definitions. It ignores parameters and automatics.

The __align specifier cannot be used on individual elements within an aggregate definition, but it can be used on an aggregate definition nested within another aggregate definition.

The __align specifier cannot be used in the following situations:

- Individual elements within an aggregate definition
- Variables declared with incomplete type
- Aggregates declared without definition
- Individual elements of an array
- Other types of declarations or definitions, such as typedef, function, and enum
- Where the size of variable alignment is smaller than the size of type alignment

For example:

Applying __align to first-level variables:

```
int __align(1024) varA;       /* varA is aligned on a 1024-byte boundary */
                              /* and padded with 1020 bytes */
static int __align(512) varB; /* varB is aligned on a 512-byte boundary */
                              /* and padded with 508 bytes */
int __align(128) functionB( );/* An error */
typedef int __align(128) T;   /* An error */
__align enum C {a, b, c};     /* An error */
```

Applying __align to align and pad aggregate tags without affecting aggregate members:

```
__align(1024) struct structA {int i; int j;}; /* structA is aligned on a
1024-byte boundary with size including padding of 1024 bytes */
__align(1024) union unionA {int i; int j;}; /* unionA is aligned on a
1024-byte boundary with size including padding of 1024 bytes */
```

Applying __align to a structure or union where the size and alignment of the aggregate using the structure or union is affected:

```
__align(128) struct S {int i;};  /* sizeof(struct S) == 128 */
struct S sarray[10];          /* sarray is aligned on 128-byte boundary */
                              /* with sizeof(sarray) == 1280 */
struct S __align(64) svar;    /* error - alignment of variable is less */
                              /* than alignment of type */
struct X {struct S s1; int a;} x;/* x is aligned on 128-byte boundary */
                                 /* with sizeof(x) == 256 bytes */
```

Applying __align to an array:

```
AnyType __align(64) arrayA[10];
/* Only arrayA is aligned on a 64-byte boundary, and elements within that
array are aligned according to the alignment of AnyType.
Padding is applied after the back of the array and does not affect the size
of the array member itself. */
```

Applying __align when the size of variable alignment differs from size of type alignment:

```
__align(64) struct S {int i;};
struct S __align(32) s1;      /* error, alignment of variable is */
                              /* smaller than alignment of type */
struct S __align(128) s2;     /* s2 is aligned on 128-byte boundary */
struct S __align(16) s3[10];  /* error */
int __align(1) s4;            /* error */
__align(1) struct S {int i;}; /* error */
```

3.10.8 Data inflation

Migrating from a 32-bit model to a 64-bit model running on a 64-bit platform also means a larger memory footprint and larger storage requirements. The change in the size of machine instructions is inevitable. The size of data items, on the contrary, is somewhat controllable in well-designed code.

3.10.8.1 Structure padding

Reordering the fields in data structures may help reduce the data inflation on 64-bit platforms. See Section 3.10.3, “Data reordering” on page 67 for an example of field rearrangement.

3.10.8.2 Cardinality - Range of possible values

Cardinality is the set of all possible values of a specific data item. For example, the cardinality of a *day-of-week* field would be [1,2,3,4,5,6,7] or [1..7]. Another example is car mileage. Four digits are used for trip mileage, counting from 0 to 9999 miles, so its cardinality is [0..9999]. For total mileage, however, six digits are used, counting from 0 to 999,999 miles. Its cardinality is [0..999999].

Now, suppose some code contains a variable of type long. What should be the type of this variable after porting from a 32-bit to a 64-bit model? Should it be changed to int in order to maintain its size? Obviously, you have to find out what kind of data this variable contains. If the variable is used to count the light years a starship can go in one mission, which is, say, 1 to 200,000 light years, it would be acceptable to change the variable type to int. On the other hand, if the variable counts in miles, which can be anything between 1 and 261 miles, it might be proper to leave the variable as type long.

3.10.8.3 Use offset instead of pointer

You do not always have to use pointers to deal with addressing. Often, using an offset or displacement to a base pointer is good enough. The advantage of using an offset value over using a pointer is that pointers are always 64 bits while the offsets can be either 2, 4, or 8 bytes, depending on their cardinality or addressing range.

3.10.8.4 Always use sizeof()

Always use the sizeof() operator to determine the actual size of a structure. Do not assume the size of the structure is just the accumulated sum of all member sizes, as additional space might be used for padding.

3.11 Lack of function prototypes

Passing arguments to a function is essentially the assignment of values to the formal parameters of the called function. For calls to functions with a prototyped declaration in scope, these assignments have implicit conversions, where argument types differ from the corresponding formal parameter type. For calls to functions lacking a prototyped declaration in scope, default argument promotions are performed on each argument. For integer data types that are 32 bits or less in size, the integral promotions will yield 32-bit int or unsigned int types. If these 32-bit values are used as 64-bit values by the called function, the behavior, according to the ANSI/ISO C standard, is undefined. This applies to both ILP32 and LP64 programming models.

The 64-bit environment calling conventions state that integral scalar parameters smaller than 64 bits are placed in the least significant bits of a 64-bit argument slot, padded on the left; the contents of the padding are undefined. Passing a non-64-bit value to a function that will use the information as a 64-bit data type will result in using undefined bits. While some versions of the C/C++ compiler, particularly with optimization disabled, may sign or zero extend arguments to 64 bits, this behavior is not guaranteed.

A similar problem will occur with a function returning a 64-bit value and no prototyped function declaration visible at the point of call. The implicit return type is int, and the callee will only expect a 32-bit value from the function called. The high order 32 bits of the return value are truncated. Note that use of implicit types is non-standard for C++, but they are being considered for the C language by the ANSI/ISO standardization committees.

3.11.1 Lack of prototyped function declaration

The examples in Figure 21 on page 76 and Figure 22 on page 77 illustrates various forms of external function declarations that appear in existing code, from non-existent to prototyped. The calls to functions func1() and func2() exhibit both problems in the code example with two separate C files (funcproto1.c and funcproto2.c in Figure 21 on page 76 and Figure 22 on page 77):

- Only 32-bit values are passed as arguments; the high order 32 bits of the argument are undefined. This occurs in lines 13 and 15 in Figure 21 on page 76.
- The call to function func2_ANSI() in Figure 21 on page 76 in line 16 also assumes an implicit `int` return types as the two previous calls (lines 13

and 15). Therefore, only 32 bits of the return value are used following the function call. This occurs in lines 13, 15, and 16 in Figure 21.

- The call to function `func3()` in Figure 21 in line 18 in the presence of a fully prototyped function declaration correctly passes a sign extended 64-bit argument and handles a 64-bit return value.

```
01  extern func2_KandR();    /* K&R function prototype */
02  extern func2_ANSI(long); /* ANSI func prototype, implicit return */
03
04  extern long func3(long);
05
06  long l1, l2, l3;
07
08  int main(int argc, char *argv[])
09  {
10    short s;
11    int i;
12
13    l1 = func1(s);        /* no function declaration visible */
14
15    l2 = func2_KandR(i);  /* implicit return type, K&R type call */
16    l2 = func2_ANSI(i);   /* implicit return type */
17
18    l3 = func3(i);
19  }
```

Figure 21. Code example funcproto1.c

```
extern long l1, l2, l3;

long func1(arg)
  long arg;
{
  return(arg * l1);
}

long func2_KandR(arg)
  long arg;
{
  return(arg * l2);
}

long func2_ANSI(long arg)
{
  return(arg * l2);
}

long func3(long arg)
{
  return(arg * l3);
}
```

Figure 22. Code example funcproto2.c

3.11.1.1 Recommendation

Prototyped function declarations should be visible at all call sites, particularly for functions with 64-bit parameters or 64-bit return types. This applies to the ILP32 mode as well as the LP64 mode. Both the compiler and `lint` should be used to locate all places where:

- Functions appear to be declared implicitly
- Functions are declared with an "old-style" parameter list
- Functions appear to have an implicit return type
- `lint` reports that function types or arguments appear to be declared or used inconsistently across source files

This combination will clearly locate problems in K&R or ANSI C source code, and, once corrected for LP64 mode, will also work in ILP32.

3.11.1.2 lint assistance

If `lint` is run on all source files that make up a binary, it will flag:

- Implicitly declared functions (at the point of call):

```
“funcproto1.c” line 13: warning: function prototype not in scope
“funcproto1.c” line 15: warning: function prototype not in scope
```

- Functions declarations with "old-style" parameter lists:

```
"funcproto2.c", line 3: warning: old style argument declaration
"funcproto2.c", line 9: warning: old style argument declaration
```

- Functions with an implicit return type of int:

```
“funcproto1.c” line 13: warning: function func1 return value used, but
none returned
“funcproto1.c” line 15: warning: function func2_KandR return value used,
but none returned
“funcproto1.c” line 16: warning: function func2_ANSI return value used,
but none returned
```

- Function arguments used inconsistently:

```
"funcproto1.c", line 16: warning: function func2_ANSI argument type
inconsistent
```

3.11.2 Pointer return or argument types without function prototype

This is a specific form of the porting issues that deal with 64-bit values used as function parameters and function return types in the absence of a prototyped function declaration in scope. In LP64 mode, a NULL pointer value (integer constant zero) used as an argument to a function without a prototyped function declaration may be passed only as a 32-bit zero, with the high order 32 bits being undefined. Likewise, in LP64 mode, a 64-bit pointer returned by a function to a callee that does not have a prototyped function declaration in scope will be treated as an int and truncated to 32 bits.

3.12 Data type promotion

Data type promotion is the conversion of operands with different data types to compatible types for comparison and arithmetic operations. For example, when a short is compared to an int, the short is first converted to an int.

Certain data type promotions, however, can result in signed numbers being treated as unsigned numbers. This, of course, may sometimes yield unexpected results.

3.12.1 Sign extension

Sign extension is a phenomenon that occurs quite often when converting to 64-bit environments. It is sometimes hard to detect, as it might have gone unnoticed before when the program suddenly seems to produce strange results. Furthermore, the integer type conversion and promotion rules are somewhat obscure. Sign extension problems can be fixed by using explicit casts to obtain the intended result.

To better understand the occurrence of sign extension, it helps to understand the integer conversion rules for ANSI C (see Section 3.5, “ANSI C integer conversion rules” on page 31).

As an example for sign extension, consider the code example in Figure 23.

```
01  #include <stdio.h>
02
03  int main(int argc, char *argv[])
04  {
05    int baseaddr;
06    unsigned long memaddr;
07
08    baseaddr = 0x10000;
09
10    memaddr = baseaddr << 15;  /* sign extension here */
11    printf("memaddr: 0x%lx %lu\n", memaddr, memaddr);
12
13    memaddr = (unsigned int)(baseaddr << 15);  /* no sign extension */
14    printf("memaddr: 0x%lx %lu\n", memaddr, memaddr);
15  }
```

Figure 23. Code example signext.c to demonstrate sign extension in LP64 mode

When compiled as a 32-bit program (in ILP32 mode), no sign extension occurs:

```
% cc -q32 -o signext32 signext.c
% ./signext32
memaddr: 0x80000000 2147483648
memaddr: 0x80000000 2147483648
%
```

However, when compiled as a 64-bit program (in LP64 mode), the variable memaddr becomes sign-extended (in line 10):

```
% cc -q64 -o signext64 signext.c
% ./signext64
memaddr: 0xffffffff80000000 18446744071562067968
memaddr: 0x80000000 2147483648
```

The sign extension occurs because the above conversion rules are applied as follows:

1. The expression baseaddr << 15 is of type int, but no sign extension has yet occurred.
2. The expression baseaddr << 15 is of type int, but it is converted to a long (See Rule 2.b in Section 3.5, "ANSI C integer conversion rules" on page 31) and then to an unsigned long before being assigned to baseaddr, because of the signed and unsigned integer promotion rule. The sign extension occurs when it is converted from an int to a long.

3.12.1.1 Recommendation

Expressions can sometimes be very tricky because of the conversion rules. Therefore, you should be very explicit about how you want the expression to be evaluated by adding casts wherever necessary.

3.12.1.2 Compiler and lint assistance

`lint` will help you find possible occurrences of sign extension. Here is the relevant output:

```
line 10: warning: conversion to long may sign-extend incorrectly
```

The VisualAge C compiler will help you find possible occurrences of sign extension when invoked with the -qwarn64 option. Here is the relevant output:

```
line 10.22: 64-bit portability: possible change of result through
conversion of int type into unsigned long int type.
```

3.12.2 Arithmetic between signed and unsigned numbers

Due to the different byte sizes of some data types, certain data promotions behave differently in LP64 mode than in ILP32 mode. This is the case, for example, when an int is compared with an unsigned long, and when an unsigned int is compared with a long.

In order to circumvent any unintended data promotion problems, programs which adhere to the ANSI C standard should perform comparisons and arithmetic operations only when all operands are either of signed type or unsigned type, as they might otherwise yield unexpected results. Consider the code example in Figure 24 on page 81.

```
01  #include <stdio.h>
02
03  int main(int argc, char *argv[])
04  {
05             int   i = -1;
06    unsigned int  ui =  1;
07             long  l = -1;
08    unsigned long ul =  1;
09
10    printf("%d < %u is %s\n",   i, ui, i < ui ? "TRUE" : "FALSE");
11    printf("%ld < %lu is %s\n", l, ul, l < ul ? "TRUE" : "FALSE");
12    printf("%d < %lu is %s\n",  i, ul, i < ul ? "TRUE" : "FALSE");
13    printf("%ld < %u is %s\n",  l, ui, l < ui ? "TRUE" : "FALSE");
14    printf("---------------\n");
15   printf("%d < %d is %s\n",   i, ui, i <  (int) ui ? "TRUE" : "FALSE")
16   printf("%ld < %ld is %s\n", l, ul, l < (long) ul ? "TRUE" : "FALSE")
17   printf("%d < %d is %s\n",   i, ul, i <  (int) ul ? "TRUE" : "FALSE")
18   printf("%ld < %ld is %s\n", l, ui, l < (long) ui ? "TRUE" : "FALSE")
19  }
```

Figure 24. Code example showing comparisons

The output of the code example above when compiled in 32-bit ILP32 mode and 64-bit LP64 mode, is shown in Table 14.

Table 14. Output of code example in Figure 24 for ILP32 and LP64 modes

Output in ILP32 mode	**Output in LP64 mode**
-1 < 1 is FALSE	-1 < 1 is FALSE
-1 < 1 is FALSE	-1 < 1 is FALSE
-1 < 1 is FALSE	-1 < 1 is FALSE
-1 < 1 is FALSE	-1 < 1 is TRUE
---------------	---------------
-1 < 1 is TRUE	-1 < 1 is TRUE
-1 < 1 is TRUE	-1 < 1 is TRUE
-1 < 1 is TRUE	-1 < 1 is TRUE
-1 < 1 is TRUE	-1 < 1 is TRUE

Clearly, the comparisons in lines 10 through 13 produce incorrect results for ILP32 mode, while in the LP64 mode the lines 10 through 12 produce incorrect results. The reason for this is that comparisons between signed and unsigned data types often do not produce the expected results, due to data type promotion. The reasons are:

1. In line 10, i is promoted to unsigned int with a value of 4,294,967,295 (for both ILP32 mode and LP64 mode) before being compared to ui.

2. In line 11, l is promoted to unsigned long with a value of 4,294,967,295 (in ILP32 mode) or a value of 18,446,744,073,709,551,615 (in LP64 mode) before being compared to ul.
3. In line 12, i is promoted to long with a value of -1 and then the same as in 2.) happens.
4. In line 13 in ILP32 mode, the long value of l=-1 is promoted to an unsigned 32-bit number with a value of 4,294,967,295. In LP64 mode, however, ui is promoted from unsigned int to long; thus, both operands are signed 64-bit numbers. This is the reason why the output of line 13 in LP64 mode is correct.

3.12.2.1 Recommendation

The code example shows the importance of ensuring that comparisons and arithmetic operations are performed only with operands of the same signed or unsigned data type (to avoid any unintended data type promotion). This is especially true when comparing an int with a long in LP64 mode.

The solution to obtain correct results is to introduce explicit casts, so that all comparisons are done with signed data types. This is shown in lines 15 through 18.

3.12.2.2 lint assistance

`lint` will only partially help you in detecting these kind of errors. The relevant `lint` output is:

```
line 12: warning: conversion to long may sign-extend incorrectly
line 17: warning: conversion from long may lose accuracy
```

The warning in line 17 is due to the fact that because of the cast from unsigned long to int, precision might be lost. In that case, casts to long could be applied to both operands.

Chapter 4. Setting up the development environment

This chapter describes how you can set up a development environment.

4.1 Your development environment

One thing that normally is *very* important to the majority of developers who use a UNIX platform, is to have access to the same or similar set of tools that they are used to working with. These tools will normally include compilers, debuggers, shells, pagers (such as more), and editors.

4.2 Online documentation

In the past, you might have left your UNIX manuals at home, or the new trainee at work might have borrowed your prized C compiler documentation. Or perhaps you are located at a customer site, where they do not use hardcopy manuals. With AIX 5L, however, all documentation is available online and is readily available for reference or selective printing. The places on the Internet where you get the appropriate documentation are described in this section.

4.2.1 AIX 5L online documentation

The AIX 5L documentation can be found online at:

`http://www.ibm-1.com/servers/aix/library/index.html`

under the section Technical Publications.

4.2.2 Compiler product information

The latest compiler products both have support Web sites that contain useful hints, tips, frequently asked questions, and links to other useful Web sites. The support page for the VisualAge C++ Professional for AIX 5L Version 5 compiler is:

`http://www.ibm.com/software/ad/vacpp/support.html`

The support page for the C for AIX Version 5 compiler is:

`http://www.ibm.com/software/ad/caix/support.html`

4.2.3 PartnerWorld for Developers

PartnerWorld for Developers is a worldwide program supporting developers who build solutions using IBM technologies. The program covers all IBM platforms, not just AIX. Its Web site contains a lot of useful information for the AIX developer, including white papers, sample code, and technology articles. It can be located on the Web at the following URL:

```
http://www.developer.ibm.com
```

4.3 Installing software on AIX

There are several different ways to install software on AIX 5L, all depending on if you want to use a graphical interface, a menu based interface, or a Command Line Interface (CLI).

4.3.1 Installing software using Web-based System Manager

If your system has a graphical user interface, the filesets can be installed using the `wsm` command. The procedure is as follows:

1. Log in as the root user.
2. Insert the product CD in the CD-ROM drive if the software was supplied on a CD.
3. Start the software installation task guide with the following command:

   ```
   # wsm install
   ```

4. From the Software drop-down menu, select **New Software (Install/Update) > Install Additional Software (Custom) > Advanced Method**.
5. In the Install Additional Software dialog, select the CD-ROM device as the software source; however, if you have downloaded your software from the Internet, or otherwise have it in a directory, then enter the path to the directory here. Then select To install specific software available from the software source.
6. Click the Browse button to generate a list of software on the media.
7. Select the desired filesets from the dialog. Click and hold down the Control button while pressing the mouse button to select one or more additional objects.
8. Click the OK button once you have selected the desired filesets to return to the Software Install dialog.
9. Click the OK button to start the install.
10. Click the Yes button to continue with the install. A pop-up panel will appear and show the output of the installation process.
11. Click the Close button once the installation has completed.

4.3.2 Installing software using SMIT

If your system does not have a graphical user interface, or you do not wish to use a Web-based System Manger, you can install the required filesets using the `smit` command as follows:

1. Log in as the root user.
2. Insert the product CD in the CD-ROM drive if the software was supplied on a CD.
3. Start the SMIT dialog with the following command:

   ```
   #smit install_latest
   ```

4. Press the F4 key to generate a list of possible input devices.
5. Select the CD-ROM device. If you have downloaded your software from the Internet, or otherwise have it in a directory, then put in the path to the directory here.
6. Press the F4 key to generate a list of available filesets.
7. Select the required filesets by highlighting them and then pressing the F7 key.
8. Press the Enter key once the required filesets have been selected.
9. Press the Enter key to start the install.
10. Press the Enter key to continue the install.
11. Press the F10 key to exit once the installation has completed.

4.3.3 Installation with the command line interface (installp)

If your system does not have a graphical user interface, or you do not wish to use a Web-based System Manger or the SMIT interface, you can install the required filesets using the `installp` command:

1. Log in as the root user.
2. Insert the product CD in the CD-ROM drive if the software was supplied on a CD.
3. Mount the CD using the command:

   ```
   # mount /<cdrom mount dir>
   ```

   If you get an error that states that /<cdrom mount dir> is not a known file system, create a CD-ROM file system using the command:

   ```
   # crfs -v cdrfs -p ro -d'cd0' -m'/mycdrom' -A'no'
   ```

   Then mount the CD using the command:

```
# mount /mycdrom
```

4. Change directory to the directory where the software you want to install is located. If you are using a CD, the install directory will normally be:

   ```
   </cdrom mount dir/usr/sys/inst.images/>
   ```

5. Now use the command `installp -ld .` to list the installable filesets on media
6. To install a fileset <fileset> in the applied state, use the command:

   ```
   installp -ad . <fileset>
   ```

   To install a fileset and commit it at the same time, use the command

   ```
   installp -acd . <fileset>
   ```

7. Use the command:

   ```
   installp -c <fileset>
   ```

   if you, at a later point in time, wish to commit an applied fileset.

Note

Some products can not be used immediately after installation. These are products that require a license. Prior to invoking such products, a product license must be enrolled with the License Use Management (LUM) system.

For a more detailed description on how to install software on AIX, see Chapter 3, “Additional software installation” in the *IBM Certification Study Guide AIX Installation and System Recovery*, SG24-6183*.* This chapter also covers how to install fixes. You should also read the manual page for the `installp` command.

4.4 The License Use Manager

IBM License Use Management Runtime, hereafter referred to as License Use Management (LUM), contains the tools needed in an end user environment to manage product licenses and get up-to-date information about license usage.

LUM is the replacement for the iFOR/LS and Net/LS systems that were used in previous versions of AIX and with previous versions of the IBM compilers.

The LUM runtime is included with AIX Version 4.3 and higher and is automatically installed. A comprehensive description of the functionality of LUM can be found in the LUM online documentation supplied on the AIX 5L product media in the ifor_ls.html.en_US.base.cli fileset. The documentation fileset is not automatically installed when installing AIX 5L; you will have to obtain your AIX installation media in order to install it.

4.4.1 Configuring LUM

After installing the LUM runtime images, one or more LUM license servers normally need to be configured. No license server needs to be configured if the licensed product supplies a simple nodelock license certificate. Both the C for AIX Version 5 and VisualAge C++ Professional for AIX Version 5 compiler products supply a simple nodelock license certificate.

The simplest method of licensing the latest compiler products is to use the simple nodelock license certificate. When this is done, there is no need to configure a LUM server; however, the installation of the certificate in large numbers of machines can be cumbersome.

If you wish to use the simple nodelock certificate, you can skip directly to Section 4.4.3, “Enrolling a product license” on page 91. If you wish to use the additional functionality available when using a license server, then the first step is to decide which server type is best suited for your environment.

There are two types of license servers:

- Concurrent nodelock license server
- Concurrent network license server

A *concurrent nodelock license server* supports concurrent nodelock product licenses. A concurrent nodelock license is local to the node where the LUM enabled product has been installed. It allows a limited number of simultaneous users to invoke the enabled licensed product on the local system.

A *concurrent network license server* supports concurrent network product licenses. A concurrent network license is a network license that can temporarily grant a user on a client system the authority to run a LUM enabled product.

Either or both of the above license servers may be configured on a single system. The number of concurrent users for the product is specified during the enrollment of the product license certificate, described in Section 4.4.3, “Enrolling a product license” on page 91.

The advantage of using a concurrent nodelock license server is that the server is installed on the same machine as the compiler and, therefore, users can obtain compiler licenses even if the machine is temporarily disconnected from the network. The disadvantage, however, is that installation of licenses is cumbersome in environments with a large number of client machines.

The main advantage of using a central network license server is that the administration of product licenses is very simple. The disadvantage is that client machines must be able to contact the license server in order to use the licensed products.

Configuring LUM requires answering several questions on how you would like to set up the LUM environment. It is recommended that users read the LUM documentation supplied with the AIX product media prior to configuring LUM.

A LUM server can be configured in several different ways. You can issue commands on the command line with appropriate arguments to configure the LUM server. You can issue a command that starts a dialog and asks a number of questions to determine the appropriate configuration, or you can configure the server using a graphical user interface.

4.4.1.1 Configuring a nodelock server

For small numbers of client machines (typically 10 or less), using a nodelock license server on each machine is the simplest method of configuring LUM. Log in as the root user and perform the following commands to configure a machine as a nodelock license server:

```
# /var/ifor/i4cfg -a n -S a
```

```
# /var/ifor/i4cfg -start
```

The first command configures the local machine as a nodelock license server and sets the option that the LUM daemons should be automatically started when the system boots. The second command starts the LUM daemons.

4.4.1.2 Using the interactive configuration tool

As an alternative to using the above commands, you can use the interactive configuration script to perform the same actions.

1. Log in as user ID root on the system where the license server will be installed.
2. Enter
   ```
   cd /var/ifor
   ```
   If this directory does not exist, then LUM has not been installed.
3. Invoke the LUM configuration tool by entering the command:

```
./i4config
```

This is the command line version of the LUM configuration tool.

4. Answer the LUM configuration questions as appropriate. The answers to the configuration questions are dependent on the LUM environment you wish to create.

   The following are typical answers to the configuration questions of LUM in order to configure both concurrent nodelock and concurrent network license servers on a single system. You may change the various answers accordingly to suit your preferred system environment. For details on configuring LUM, please read the documentation that comes with LUM.

   - Select 4 “Central Registry (and/or Network and/or Nodelock) License Server” on the first panel.
   - Answer y to “Do you want this system be a Network License Server too?”
   - Select 2 “Direct Binding only” as the mechanism to locate a license server.
   - Answer n to “Do you want to change the Network License Server ip port number?”
   - Answer n to “Do you want to change the Central Registry License Server ip port number?”
   - Answer n to “Do you want to disable remote administration of this Network License Server?”
   - Answer y to “Do you want this system be a Nodelock License Server too?”
   - Select 1 “Default” as the desired server(s) logging level.
   - Enter blank to accept the default path for the default log file(s).
   - Answer y to “Do you want to modify the list of remote License Servers this system can connect to in direct binding mode (both for administration purposes and for working as Network License Client)?”
   - Select 3 “Create a new list” to the direct binding list menu.
   - Enter the host name, without the domain, of the system you are configuring LUM for when prompted for the “Server network name(s).”
   - Answer n to “Do you want to change the default ip port number?”
   - Answer y to “Do you want the License Server(s) automatically start on this system at boot time?”

- Answer y to continue the configuration setup and write the updates to the i4ls.ini file.
- Answer y to “Do you want the License Server(s) start now?”

Both concurrent nodelock and concurrent network license servers should now be configured on your system.

For more information on configuring and using LUM, refer to the LUM documentation supplied with AIX. As an alternative, the LUM manual, *Using License Use Management Guide Runtime for AIX*, SH19-4346, can be viewed online in PDF format at the following URL:

ftp://ftp.software.ibm.com/software/lum/aix/doc/V4.5.5/lumusgaix.pdf

4.4.2 Activating the LUM server

After configuring and starting the LUM server, you can enroll product licenses. Before attempting to enroll a license, you must first ensure that the LUM daemons are active. This can be done with the following command:

```
# /var/ifor/i4cfg -list
```

Depending on the type of LUM server configured, the output will be similar to the following:

```
i4cfg Version 4.5 AIX -- LUM Configuration Tool
(c) Copyright 1995-1998, IBM Corporation, All Rights Reserved
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with IBM Corp.
Subsystem   Group    PID   Status
i4llmd      iforls   22974 active
```

If no subsystem is listed as active, then start them with the following command:

```
# /var/ifor/i4cfg -start
```

The only daemon that must be active is the Nodelock License Server Subsystem (i4llmd) daemon. The other daemons that may be active depending on your configuration are as follows:

- License Sever Subsystem (i4lmd)
- Central Registry Subsystem (i4gdb)
- Global Location Broker Data Cleaner Subsystem (i4glbcd)

4.4.3 Enrolling a product license

After LUM has been installed and configured on your system, the product license certificates can be enrolled with the LUM license server. Three LUM product license certificates are provided with each of the latest compiler products:

1. Concurrent nodelock license certificate
2. Concurrent network license certificate
3. Simple nodelock license certificate

You should enroll the appropriate license certificate for the type of LUM environment you have configured. For example, the locations of the license certificates for the compiler products are detailed in Table 15.

Table 15. License certificate locations

Compiler	License Certificate Type	Location
C for AIX Version 5	Concurrent Network	/usr/vac/cforaix_c.lic
	Concurrent Nodelock	/usr/vac/cforaix_cn.lic
	Simple Nodelock	/usr/vac/cforaix_n.lic
VisualAge C++ Professional for AIX Version 5	Concurrent Network	/usr/vacpp/vacpp_c.lic
	Concurrent Nodelock	/usr/vacpp/vacpp_cn.lic
	Simple Nodelock	/usr/vacpp/vacpp_n.lic

4.4.4 Enrolling a concurrent license

To enroll a concurrent network or concurrent nodelock license certificate, perform the following steps:

1. Log in as root on the system where the license server is installed.
2. Invoke the LUM configuration tool by entering the LUM Basic License Tool command as follows:

   ```
   /var/ifor/i4blt
   ```

The i4blt tool contains both a graphical user interface and a command line interface. Note that the LUM daemons must be running before starting the i4blt tool. Refer to Section 4.4.2, “Activating the LUM server” on page 90 for information on how to check the status of the LUM daemons.

If the X11 runtime (X11.base.rte fileset) has been installed on your system, the GUI version of the tool will be invoked. Otherwise, the command line version will be invoked, and an error will occur, because the appropriate command line parameters were not specified.

The following are the instructions for both interfaces using the i4blt tool:

- Enrolling using the graphical user interface:
  - Select the **Products** pull-down and click on the Enroll Product item.
  - Click on the Import button. The Import panel should be displayed.
  - For example, if you want to enroll a license for the Visualage C++ compiler, then enter `/usr/vacpp/*.lic` or `/usr/vac/*.lic` (if you are enrolling a license for C for AIX) in the Filter entry prompt, and press Enter. This will show the various product license files in the Files panel. The three license files for the product, as detailed in Table 15 on page 91, should be displayed.
  - Select either the prod_c.lic or prod_cn.lic (where prod is either vacpp or cforaix) license by clicking on the entry.
  - Click OK. The Enroll Product panel should be redisplayed with information regarding the product indicated.
  - Click on the OK button of the Enroll Product panel. The Enroll Licenses panel should be displayed.
  - Fill in the information on the Administrator Information portion of the panel (optional).
  - Fill in the number of valid purchased licenses of the product under Enrolled Licenses in the Product information portion of the panel (mandatory).
  - Click on the OK button of the Enroll Licenses panel. The product should be successfully enrolled. You may terminate the i4blt tool.
- Enrolling using the command line:
  - From the required product license file, as detailed in Table 15 on page 91, extract the `i4blt` command from the top of the file.
  - Replace `number_of_lics` from the command with the number of valid purchased licenses of the product (mandatory).
  - Replace `admin_name` with the name of the administrator (optional).
  - Invoke this command as root from `/var/ifor`. The product should be successfully enrolled.

4.4.5 Enrolling a simple nodelock license

Read the instructions at the top of the simple nodelock license certificate file. In general, this type of license will be installed when no LUM system has been configured. This means enrolling the license is simply a case of placing the indicated license information line into the /var/ifor/nodelock LUM nodelock file.

4.5 Shells available on AIX 5L

In a UNIX-based environment, developers would likely prefer to have a prompt, or even refuse to work without one. There are some platforms where it is possible to work entirely within a graphical interface, never actually issuing commands from a prompt. In UNIX-based environments, the prompt is provided by one of the many widely used shells.

AIX 5L comes with several shells already installed. These are:

- sh
- ksh
- csh
- bsh
- tsh
- ksh93

sh and tsh are hardlinks to ksh, which is an enhanced version of the1988 KornShell. The ksh93 is an unmodified version of the 1993 version of ksh. This version is also POSIX compliant. With the exception of POSIX specific items, the 1993 version should be backward compatible with the 1988 version. Therefore, no changes to shell scripts should be necessary. For detailed information on the ksh93, consult the official KornShell Web site at:

```
http://www.kornshell.com
```

The ksh93 is located in /usr/bin/ksh93.

Furthermore, you might read about the different shells in AIX 5L in the *AIX 5L Version 5.1 System User's Guide: Operating System and Devices* (found in the AIX 5L online documentation), which has a chapter describing the shells available in AIX 5L.

There are several other popular shells, such as bash and tcsh. These can be obtained from one of the places discussed in Section 4.8, “Where to get GNU and other useful software for AIX 5L” on page 96. If you wish to change your default login shell to, for example, tcsh, and also allow people to use the bash shell as a login shell, which you already have installed on your AIX 5L machine, you have to:

1. Log in as the root user.
2. Add the shell to the shells stanzas in the /etc/security/login.cfg file.
3. Change a user's default shell by using the `chsh` command.
4. Exit the root shell.
5. Login as the user.

Changing the login shell for the user jasper from ksh to tcsh, might be done the way it is done in Figure 25.

```
$ su - root
root's Password:
# ed /etc/security/login.cfg
4084
/shells =
        shells =
/bin/sh,/bin/bsh,/bin/csh,/bin/ksh,/bin/tsh,/bin/ksh93,/usr/bin/sh,/
usr/bin/bsh,/usr/bin/csh,/usr/bin/ksh,/usr/bin/tsh,/usr/bin/ksh93,/u
sr/sbin/sliplogin,/usr/sbin/snappd,/usr/sbin/uucp/uucico
s#uucico#uucico,/usr/local/bin/tcsh,/usr/local/bin/bash#p
        shells =
/bin/sh,/bin/bsh,/bin/csh,/bin/ksh,/bin/tsh,/bin/ksh93,/usr/bin/sh,/
usr/bin/bsh,/usr/bin/csh,/usr/bin/ksh,/usr/bin/tsh,/usr/bin/ksh93,/u
sr/sbin/sliplogin,/usr/sbin/snappd,/usr/sbin/uucp/uucico,/usr/local/
bin/tcsh,/usr/local/bin/bash
w
4124
q
# chsh jasper /usr/local/bin/tcsh
# lsuser -a shell jasper
jasper shell=/usr/local/bin/tcsh
# exit
$ exit
```

Figure 25. Changing your default login shell from ksh to /usr/local/bin/tcsh

Now when you login, you will have the tcsh as the login shell. To make the bash shell your login shell, simply substitute tcsh for bash in the example.

Note

It is *NOT* recommended to change the shell of the root user, as this might have unforeseen consequences for your system.

4.6 Editors available on AIX 5L

AIX 5L comes with several different editors; some are graphical editors and some are not. These editors are:

- dtpad
- ed
- ex
- vi

These editors should be known to most developers. There are also several freeware editors which are widely used. Two commonly used editors are:

- emacs
- nedit

emacs is perhaps the most well known editor. emacs, and the documentation for it, can be obtained from the places mentioned in Section 4.8.2, “Other locations for GNU software for AIX 5L” on page 97. If you download emacs from the Bull site, you might want to check the execute permissions on the /usr/local/bin/emacs<version> file. emacs will work whether you work in a XWindow or 80x24 ASCII environment.

nedit is an XWindow editor. nedit is short for Nirvana Editor.

4.7 Source Code Control products under AIX 5L

AIX 5L comes with the SCCS source code control utility available on the product media. In Figure 26, you can see that SCCS is installed as the bos.adt.sccs fileset, and that all the files are located in the /usr/bin. directory.

```
$ lslpp -f bos.adt.sccs
  Fileset             File
  -----------------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.adt.sccs 5.1.0.0  /usr/bin/comb
                        /usr/bin/rmdel
                        /usr/bin/val
                        /usr/bin/get
                        /usr/bin/delta
                        /usr/bin/sccs
                        /usr/bin/cdc -> /usr/bin/rmdel
                        /usr/bin/sccsdiff
                        /usr/bin/sact -> /usr/bin/unget
                        /usr/bin/vc
                        /usr/bin/admin
                        /usr/bin/unget
                        /usr/bin/sccshelp
                        /usr/bin/prs
$
```

Figure 26. The Source Code Control System

To invoke help on SCCS, use the `sccshelp` command.

Besides several commercial Source Code Control products, another widely used product is RCS, which is short for Revision Control System. If you are using RCS and want to continue to use RCS, the GNU version can be obtained from the places mentioned in Section 4.8.2, “Other locations for GNU software for AIX 5L” on page 97. The `make` command on AIX 5L does not, by default, support RCS. If you are currently using the GNU make and RCS combination, then it might be a good idea to continue to do that.

4.8 Where to get GNU and other useful software for AIX 5L

There are several places where you can obtain or order GNU and other freeware software. In this section, we will try to list some of the best places to get hold of freeware software.

4.8.1 AIX Toolbox for Linux Applications

The AIX Toolbox for Linux Applications CD is shipped with AIX 5L. The CD contains a collection of popular open source and GNU software built for AIX 5L. The CD is shipped with AIX 5L media. If you can not locate your Toolbox CD, the images can be downloaded from the AIX Toolbox web site at:

`http://www.ibm.com/servers/aix/products/aixos/linux/`

New software and new versions of existing software will become available on a regular basis.

4.8.2 Other locations for GNU software for AIX 5L

If the piece of software you want to use is not available on the AIX Toolbox CD or Web site, there are several other places where you can download them from. Some of the GNU software used in this book was downloaded from Bull's large Freeware and Shareware Archive for AIX 4, which can be found at the following URL:

`http://www-frec.bull.com/pub`

With mirror sites in Europe

`http://ftp.univie.ac.at/aix/`

And mirror sites in the United States

```
http://www.rge.com/pub/systems/aix/bull/
ftp://ftp.rge.com/pub/systems/aix/bull/
```

The practical thing about downloading from this site is that all the software is already packed in installp format packages for easy installation and maintenance.

If you want to get GNU software source code or just want to compile it yourself, you can use the GNU Web site:

`http://www.gnu.org/`

The site also contains documentation on the various GNU software products.

4.8.3 Downloading Nedit for AIX 5L

Nedit for AIX 5L and the source code can be downloaded from the Bull Web page described above, or from the Nedit home page:

`http://www.au.nedit.org/`

The documentation is also available, together with the source code, from this Web site.

4.9 Compilers available on AIX 5L for Power

The IBM C and C++ compiler products for AIX 5L share some similar characteristics, such as the way the products are installed on the system and the configuration options available when using the products.

4.9.1 IBM C for AIX Version 5.0.2

The C for AIX Version 5.0.2 compiler is the latest IBM C compiler product available for AIX. It extends the existing symmetric multi-processing (SMP) support available with C for AIX Version 4.4 by supporting the OpenMP industry specification. OpenMP provides a model for parallel programming that allows a program to be portable across shared memory architectures from different vendors by using a common set of application program interfaces. The compiler generates highly-optimized code for all RS/6000 processors and can provide run-time address checking to detect memory errors.

This compiler is only supported by IBM AIX Version 4.2.1 or later. Also, note that 64-bit applications will run only on AIX Version 4.3 and later when running on 64-bit hardware.

C programs written using Version 3 or Version 4 of IBM C for AIX are source compatible with IBM C for AIX Version 5.0. C programs written using either Version 2 or 3 of IBM Set ++ for AIX or the XL C compiler component of AIX Version 3.2 are source compatible with IBM C for AIX Version 5.0, with exceptions to detect invalid programs or areas where results are undefined.

This version of the compiler is installed under /usr/vac and uses the /etc/vac.cfg configuration file. If C for AIX Version 4.x is installed on a system, installing C for AIX Version 5.0.2 will overwrite and upgrade the previous version.

The C for AIX Version 5.0.2 compiler uses the LUM licensing system to control usage of the product. Refer to Section 4.9.6, “Activating the IBM compilers” on page 103 for information on configuring the license system.

4.9.2 IBM VisualAge C++ Professional for AIX Version 5.0.2

VisualAge C++ Version 5.0.2 features a fully incremental compiler and a new batch compiler. The Integrated Development Environment (IDE) operates with the incremental compiler when used in the AIX Common Desktop Environment (CDE). The batch compiler is run from the command line and is suitable for use in a development environment that uses makefiles. Both compilers support the latest ANSI/ISO C++ language standard and the latest version (Version 5) of the IBM Open Class library.

The main differences between Version 4 and Version 5 of this product are:

- Version 5 supports multiple codestores in a single project.
- Version 5 is a single product featuring both batch and incremental compilers.

The graphical interface of Version 5 has been redesigned with a host of helpful features. Version 5 has improved optimization techniques and provides the programmer with effective and efficient ways of handling C++ object code. Also, this product allows the developer to carry out performance analysis to determine the applications usage of system resources.

This product is supported on IBM AIX Version 4.2.1 and later versions for RS/6000 hardware.

As described above, this version of VisualAge features an incremental compiler. The implications of this for productivity and the code are impressive, but if the application is moving from a batch environment, do spend time with the application to adapt to the VisualAge products. For example, makefiles cannot be processed directly by the incremental compiler.

But, once the migration is done, then the advantages of VisualAge products are very impressive. This then would reduce the amount of time and memory required to do each build, as well as the time spent on rebuilding when some changes are made to the source files.

C++ programs written using Version 4 of IBM VisualAge Professional for AIX, and IBM C and C++ compilers, Version 3.6 and earlier, are source compatible with the VisualAge C++ Professional for AIX Version 5. Since the product features a batch compiler in addition to the incremental compiler, there are situations where one is more suitable than the other.

4.9.3 Multiple command line drivers

Each compiler product, with the exception of VisualAge C++ Version 4.0, has multiple command line driver interfaces available, each causing a different set of default arguments to be used. For example, the C compiler products provide commands, such as `cc`, `xlc`, `c89`, `cc_r`, and so on. These commands are all links to a single compiler core, which uses a specific set of options, depending on the name of the command used to invoke it.

In addition to the default invocation commands provided when the compiler is installed, the system administrator can create new commands, which result in the compiler being invoked with a customized set of default options. This feature is controlled by the compiler configuration file, which lists the options to be used for each invocation command. The exact name of the configuration file differs between the compiler products, but generally has a name of the form /etc/*comp.cfg*, where *comp* indicates the compiler product that uses the configuration file.

4.9.3.1 Finding the compiler drivers

The earlier versions of the compiler products automatically created symbolic links in /usr/bin for each invocation command supplied by the compiler. For example, this means that if a user has the directory /usr/bin as part of their PATH environment variable (which it is by default), they need only type `cc` on the command line to invoke the `/usr/bin/cc` command.

The later versions of the compiler products are designed to co-exist with earlier versions, and, as a consequence, they do not create the symbolic links in /usr/bin when they are installed. This means that a user may have trouble invoking the compiler on a system that only has a new version compiler product installed. There are two solutions available in this instance:

- When logged in as the root user, invoke the `replaceCSET` command supplied with the compiler. This will create appropriate symbolic links in /usr/bin to the compiler driver programs.
- Alter the PATH environment variable to add the directory that contains the compiler driver programs. For example:

  ```
  PATH=/usr/vac/bin:$PATH; export PATH
  ```

The second solution should be used if two compilers are installed on a system, because it allows each user to choose which version of the compiler they wish to use. If the system only has one compiler installed, it makes sense to use the first solution. If required, the root user can reverse the action of the `replaceCSET` command by using the `restoreCSET` command, which is also supplied with the compiler. The exact location of the `replaceCSET` and `restoreCSET` commands will depend on the version of the compiler you are using.

4.9.4 Installation directory

The main components of the compiler product are installed on the system in the /usr file system. The exact directory used depends on the compiler product. For the C compiler, the directory is /usr/vac and for the C++ compiler, the directory is /usr/vacpp.

4.9.5 Installation of compiler products

The installation of the latest compiler products (C for AIX Version 5 and VisualAge C++ Professional for AIX Version 5) is a very simple task. There are a number of steps that need to be performed to end up with correctly installed and working compilers.

The first step in the installation process is to install the compiler product filesets onto the system. The filesets to be installed will vary, depending on the compiler product and the desired configuration.

4.9.5.1 Selecting required filesets

The compiler products are delivered on CD-ROM media and are accompanied with a license certificate for the number of licenses purchased. The CD-ROM media includes the compiler filesets along with a number of other filesets, some of which are optionally installable, and some of which are co-requisites of the compiler filesets and are automatically installed. Table 16 on page 101 lists the main packages on the C for AIX Version 5 CD-ROM, and Table 17 lists the main packages on the VisualAge C++ Professional for AIX Version 5 CD-ROM media.

Table 16. C for AIX Version 5 packages

Package Name	Description
IMNSearch	Search engine for HTML documentation
idebug	Debugger with graphical user interface
memdbg	Memory debugging tools
vac	C compiler
xlC	C++ library (required by compiler executables)
xlsmp	Parallelization run-time component

Table 17. VisualAge C++ Professional for AIX Version 5 packages

Package Name	Description
IMNSearch	Search engine for HTML documentation
idebug	Debugger with graphical user interface
ipfx	Information presentation tool (used for viewing manuals)
memdbg	Memory debugging tools
vac	C compiler
vacpp.Dt	Desktop integration
vacpp.cmp.batch	Batch (command line) C++ compiler
vacpp.cmp. incremental	Incremental C++ compiler
vacpp.cmp.C	C compiler integration

Package Name	Description
vacpp.dax	Data access builder
vacpp.ioc	IBM Open Class Library
vacpp.lic	License files
vacpp.memdbg	C++ memory debugging tools
vacpp.rescmp	Resource compiler
vacpp.vb	Visual Builder
vatools	Additional C++ development tools
xlC.adt	Additional C++ header files

In all cases, the target AIX system should already have the bos.adt.include fileset installed, which contains the system provided header files. The other filesets in the bos.adt package contain useful tools and utilities often used during application development, so it is a good idea to install the entire package. Neither the bos.adt package or bos.adt.include fileset is installed by default when installing AIX on a machine. If your system does not have the filesets installed, you will need to locate your AIX installation media and install them prior to installing the compilers, because these filesets are AIX version specific and are not supplied on the compiler CD-ROM product media.

When installing the C for AIX Version 5 product, installing the vac.C fileset will automatically install the minimum of additional required filesets. The additional filesets you may wish to install are the documentation filesets.

When installing the VisualAge C++ Professional for AIX Version 5 product, your choice of filesets will depend on whether you wish to install the batch (command line) C++ compiler, incremental C++ compiler, C compiler, or a combination of the three.

For simple C++ command line compiles, installing the vacpp.cmp.batch fileset will automatically include the minimum required filesets. Additional filesets can be selected, depending on the type of development work being done, such as vacpp.vb for installing the components used for building applications using the Visual Builder component.

Note

Regardless of whether you are using the incremental of batch compiler, ensure that the vacpp.lic fileset is installed, as this contains the license files required when activating the compiler.

Regardless of the product or required configuration, the filesets can be installed using one of the methods discussed in Section 4.3, “Installing software on AIX” on page 84.

4.9.6 Activating the IBM compilers

Once you have installed the desired compiler filesets onto the system, the next step in the process is to enroll a license for the product into the LUM system. Section 4.4, “The License Use Manager” on page 86 describes the process of configuring a LUM server and enrolling a product license. If you already have a LUM environment enabled, you may go directly to Section 4.4.3, “Enrolling a product license” on page 91.

4.10 Invoking the IBM compilers

Once a compiler product license has been enrolled, you are now ready to use the compilers. As mentioned in Section 4.9.4, “Installation directory” on page 100, the compiler drivers are not installed in a directory that is searched with the default PATH environment variable. There are a number of methods of resolving this issue:

- If you do not have a previous version of the compiler installed, then, as the root user, invoke the replaceCSET script installed with the compiler. It will be in the /usr/vac/bin directory.
- Add the directory containing the compiler drivers to the default PATH environment variable set in the /etc/environment configuration file.
- Add the directory containing the compiler drivers to the PATH environment variable in each users’ .profile shell configuration file.
- Change the makefiles used in your development environment to configure the compiler macro to use the absolute path. For example:

  ```
  CC=/usr/vac/bin/cc
  ```

Using the replaceCSET script is the preferred option, because it resolves the problem for all users after a simple single action by the root user.

4.10.1 Default compiler drivers

The Version 5 compiler products include a number of default compiler configurations in the /etc/vac.cfg compiler configuration file. The default C++ command line driver is /usr/vacpp/bin/xlC. The three main C compiler command line drivers are as follows:

/usr/vac/bin/cc	Extended mode C compiler
/usr/vac/bin/xlc	ANSI C compiler, using UNIX header files
/usr/vac/bin/c89	ANSI C compiler, using ANSI C header files

There are a number of additional command line drivers available, each one based on the basic cc, xlc and xlC drivers described above. They are described in Table 18.

Table 18. Compiler driver extensions

Package Name	Description
_r	Uses the UNIX 98 threads libraries.
_r7	Uses the POSIX Draft 7 threads libraries.
_r4	Uses the POSIX Draft 4 (DCE) threads libraries.
128	Enables 128 bit double precision floating point values and uses appropriate libraries.
128_r	Enables 128 bit double precision floating point values and uses the UNIX 98 threads libraries.
128_r7	Enables 128 bit double precision floating point values and uses the POSIX Draft 7 threads libraries.
128_r4	Enables 128 bit double precision floating point values and uses the POSIX Draft 4 (DCE) threads libraries.

For example, to compile an ANSI C program using Draft 7 of the POSIX threads standard, use the xlc_r7 compiler driver. To compile a C++ program that uses 128 bit floating point values, use the xlC128 compiler driver.

4.11 Online compiler documentation

The Version 5 compilers come with online documentation that is written in HTML format. The default configuration makes it very easy to view the online documentation on the machine on which it is installed.

4.11.1 Viewing locally

The procedure for viewing the documentation installed on the local machine depends on a number of factors, including which compiler product is installed and whether you are using the AIX Common Desktop Environment.

4.11.1.1 C compiler documentation

The C for AIX Version 5 compiler documentation is written in HTML format. The HTML files are located in the /usr/vac/html directory. To view the documentation, start the Netscape browser supplied with the AIX Bonus Pack and point it at the following file:

```
/usr/vac/html/en_US/doc/index.htm
```

Before starting Netscape, ensure that the environment variable SOCKS_NS is not set. For the search facility to work correctly, the browser must not have proxy handling enabled for the localhost port. To disable proxy handling for the local host when using Netscape, do the following:

1. Start the browser, then select **Edit->Preferences** from the menu.
2. Double-click Advanced in the Category tree.
3. Click Proxies in the Advanced subtree.
4. Click View at the Manual Proxy Configuration selection.
5. Type the following in the “Do not use proxy servers for domains beginning with:” box:

   ```
   localhost:49213
   ```

   If there are other entries in the box, separate the new entry with a comma.
6. Click OK, then click OK to exit the Preferences panel.

4.11.1.2 C++ compiler documentation

The VisualAge C++ Professional for AIX Version 5 compiler documentation is written in HTML format. The HTML files are stored in a single file in ZIP format. The files are viewed using an HTML browser, which uses a cgi-bin script to extract and view the required files. There is no need to manually unpack the ZIP file.

If you are using the AIX CDE interface, the C++ compiler documentation can be started by double-clicking on the Help Homepage icon in the VisualAge C++ Professional folder of the Application Manager.

If you are not using the AIX CDE interface, or are logged in remotely from another X11 capable display, then use the following command:

```
/usr/vacpp/bin/vacpphelp
```

The command starts the default Netscape browser (which is supplied on the AIX Bonus Pack media) with the correct URL.

4.11.2 Viewing remotely

By default, it is not possible to view the online documentation from a remote machine. It can be done in a simple way by logging in to the machine that has the documentation installed, setting the DISPLAY environment variable to use a remote X11 display, then viewing the documentation by invoking the same command used to view locally.

A better solution, particularly in larger environments or where remote clients do not have X11 capabilities, is to configure the machine to allow remote viewing of the documentation. This can be performed as shown in the following sections.

4.11.2.1 Configuring the HTTP server

Suppose the machine that has the documentation filesets installed has a fully qualified domain name of `docs.ibm.com`. The following example demonstrates the steps performed on that machine to allow remote clients to view the compiler documentation using their HTML browser:

1. Log in as the root user.
2. Perform the following command:

   ```
   cp /etc/IMNSearch/httpdlite/httpdlite.conf
   /etc/IMNSearch/httpdlite/vacpp.conf
   ```

3. Edit `/etc/IMNSearch/httpdlite/vacpp.conf`, and make the following changes:

   a. Change the HostName line from:

      ```
      HostName localhost
      ```

      to:

      ```
      HostName docs.ibm.com
      ```

      If the HostName line is not present, or has a comment symbol (`#`) at the start of the line, then simply add the following line to the file:

      ```
      HostName docs.ibm.com
      ```

   b. Change the Port line from:

      ```
      Port 49213
      ```

      to:

```
Port 49214
```

c. If the version of `IMNSearch.rte.httpdlite` installed on your machine is greater than 2.0.0.0, you will need to add one or more Allow lines to specify which hosts are permitted to access the Web server. The Allow statement has the following syntax:

```
Allow network-ip network-mask
```

A client is only granted access if the following rule is met: (& is a bitwise AND operation)

```
client-ip & network-mask == network-ip & network-mask
```

For example, if you wanted machines with an address, such as 9.x.x.x, to be able to access the help server, you would add the following statement to vacpp.conf:

```
Allow 9.0.0.0 255.0.0.0
```

d. Save the file and exit the editor.

4. Edit the file /etc/inittab. There is a line that executes the `httpdlite` command with a file name argument. The line is as follows:

```
httpdlite:2:once:/usr/IMNSearch/httpdlite/httpdlite -r
/etc/IMNSearch/httpdlite/httpdlite.conf >/dev/console 2>&1
```

Make a copy of this line immediately below the original line. In the new line:

a. Change the first field from httpdlite to httpdlite2.

b. Change the part of the line that reads httpdlite.conf to vacpp.conf.

The result should be as follows:

```
httpdlite2:2:once:/usr/IMNSearch/httpdlite/httpdlite -r
/etc/IMNSearch/httpdlite/vacpp.conf >/dev/console 2>&1
```

Save the file and exit from the editor.

5. Reboot the system or run the following command to start the second copy of the ICS lite server:

```
/usr/IMNSearch/httpdlite/httpdlite -r
/etc/IMNSearch/httpdlite/vacpp.conf >/dev/console 2>&1
```

The steps described above configure an instance of an HTTP server to respond on a specific port number to requests to access compiler documentation.

The following sections detail the additional steps required to configure the documentation for each compiler product to be served by the HTTP server.

4.11.2.2 Configuring the C++ documentation

The following steps are required to enable the online documentation for the VisualAge C++ Professional for AIX Version 5 compiler to be served by the HTTP server:

1. Log in as the root user.
2. Change the directory to /var/vacpp/en_US.
3. Edit the file hgssrch.htm and change the line:

   ```
   <form ACTION="http://localhost:49213/cgi-bin/vacsrch.exe"
   METHOD="POST">
   ```

   to:

   ```
   <form ACTION="http://docs.ibm.com:49214/cgi-bin/vacsrch.exe"
   METHOD="POST">
   ```

   Save the file and exit the editor.
4. Issue the following command:

   ```
   /usr/IMNSearch/cli/imndomap -u "VACENUS"
   "http://docs.ibm.com:49214/cgi-bin/vahwebx.exe/en_US/vacpp/Extract/0/"
   "VisualAge C++"
   ```

5. Users can point their browser at the following URL to browse and search the documentation:

   ```
   http://docs.ibm.com:49214/cgi-bin/vahwebx.exe/en_US/vacpp/Extract/0/ind
   ex.htm
   ```

4.11.2.3 Configuring the C compiler documentation

The following steps are required to enable the online documentation for the C for AIX Version 5 compiler to be served by the HTTP server:

1. Log in as the root user.
2. Change the directory to /usr/docsearch/html.
3. Perform the following command:

   ```
   ln -s /usr/vac/html/en_US/doc vac_doc
   ```

4. Edit the file /usr/vac/html/en_US/doc/hgssrch.htm and change the line:

   ```
   <form ACTION="http://localhost:49213/cgi-bin/caixsrch.exe"
   METHOD="POST">
   ```

   to:

   ```
   <form ACTION="http://docs.ibm.com:49214/cgi-bin/caixsrch.exe"
   METHOD="POST">
   ```

   Save the file and exit the editor.

5. Issue the following command:

```
/usr/IMNSearch/cli/imndomap -u "CENUS"
"http://docs.ibm.com:49214/vac_doc/" "C for AIX"
```

6. Users can point their browser at the following URL to browse and search the documentation:

```
http://docs.ibm.com:49214/vac_doc/index.htm
```

4.12 The GNU compilers

The GNU C (gcc) and C++ (g++) compilers are freeware compilers. At the time of writing this book, these compilers are available for AIX 5L for Power systems. It is anticipated that versions for AIX 5L for Itanium-based systems will be available in the near future. See Section 8.2, “GNU GCC for AIX 5L” on page 249 for more information.

4.13 The lint code checker

The `lint` command checks C and C++ language source code for coding and syntax errors and for inefficient or non-portable code. You can use this program to:

- Identify source code and library incompatibility
- Enforce type-checking rules more strictly than the compiler
- Identify potential problems with variables
- Identify potential problems with functions
- Identify problems with flow control
- Identify legal constructions that may produce errors or be inefficient
- Identify unused variable and function declarations
- Identify possibly non-portable code

Notes

Checking of C++ language files with the `lint` command requires the presence of the VisualAge C++ Professional for AIX Compiler package.

The inter-file usage of functions is checked to find functions that return values in some instances and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but not returned. The `lint` command interprets file name extensions as follows:

- File names ending in .c (small ‘c’) are C language source files.
- File names ending in .C (capital ‘C’) are C++ language source files.
- File names ending in .ln are non-ASCII files that the lint command produces when either the -c or the -o flag is used.

The `lint` command warns you about files with other suffixes and ignores them. The `lint` command takes all the .c, .C, and .ln files and the libraries specified by -l flags and processes them in the order that they appear on the command line. By default, it adds the standard llib-lc.ln lint library to the end of the list of files. However, when you select the -p flag, the `lint` command uses the llib-port.ln portable library.

By default, the second pass of `lint` checks this list of files for mutual compatibility; however, if you specify the -c flag, the .ln and llib-lx.ln files are ignored. The -c and -o flags allow for incremental use of the `lint` command on a set of C and C++ language source files. Generally, use the `lint` command once for each source file with the -c flag. Each of these runs produces a .ln file that corresponds to the .c file and writes all messages concerning that source file.

After you have run all source files separately through the `lint` command, run it once more, without the -c flag, listing all the .ln files with the needed -l flags. This determines all inter-file inconsistencies. This procedure works well with the `make` command, allowing it to run the `lint` command on only those source files modified since the last time that set of source files was checked. The lint and LINT preprocessor symbols are defined to allow certain questionable code to be altered or ignored by the `lint` command. Therefore, the lint and LINT symbols should be thought of as a reserved word for all code that is planned to be checked by `lint`.

4.14 Debuggers available on AIX 5L

Debuggers are an essential part of any development environment. For AIX 5L, debuggers are readily available.

4.14.1 Included debuggers

AIX 5L comes with two debuggers:

- adb
- dbx

where dbx is the more advanced debugger. The dbx debugger is not supported on AIX 5L for Itanium-based systems.

A detailed description of how to use these debuggers and how they work can be found in the manual pages for AIX 5L and in *General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation.

4.14.2 idebug and irmtdbgc

The `idebug` command starts both the IBM Distributed Debugger interface and the debug engine when debugging a program locally. When debugging remotely, it is used to connect to a debug engine daemon on a remote system or to start the debugger user interface as a daemon on your local system. The `idebug` debugging system is supplied on the IBM C for AIX Version 5 and IBM VisualAge C++ Professional for AIX Version 5 CD-ROM media.

The `irmtdbgc` command starts the debug engine on the remote system. If the debug engine detects a debugger user interface daemon, then you can start debugging your program immediately. If no debugger user interface daemon is detected, the debug engine will run as a daemon until you start the debugger user interface on the local system with the `idebug` command.

4.15 AIX 5L directories

The root file system is the top of the hierarchical file tree. It contains the files and directories critical for system operation, including the device directory and programs for booting the system. The root file system also contains mount points where file systems can be mounted to connect to the root file system hierarchy. The following list provides information about the contents of some of the subdirectories of the `/` (root) file system.

/etc	Contains configuration files that vary for each machine. Examples include /etc/hosts and /etc/passwd. The /etc directory contains the files generally used in system administration.
/bin	Symbolic link to the /usr/bin directory. In prior UNIX-based file systems, the /bin directory contained user commands that now reside in the /usr/bin directory.
/sbin	Contains files needed to boot the machine and mount the /usr file system. Most of the commands used during booting come from the boot image's RAM disk file system; therefore, very few commands reside in the /sbin directory.

/dev	Contains device nodes for special files for local devices. The /dev directory contains special files for tape drives, printers, disk partitions, and terminals.
/tmp	Serves as a mount point for a file system that contains system-generated as well as application-generated and user-created temporary files.
/var	Serves as a mount point for files that vary on each machine. The /var file system is configured as a file system since the files it contains tend to grow.
/u	Symbolic link to the /home directory.
/usr	Contains files that do not change and can be shared by machines, such as executables and ASCII documentation. Standalone machines mount the root of a separate local file system over the /usr directory. Diskless machines mount a directory from a remote server over the /usr directory.
/home	Serves as a mount point for a file system containing user home directories. The /home file system contains per-user files and directories. In a standalone machine, the /home directory is contained in a separate file system whose root is mounted over the root file system's /home directory. In a network, a server might contain user files that should be accessible from several machines. In this case, the server's copy of the /home directory is remotely mounted onto a local /home file system.
/export	Contains the directories and files on a server that are for remote clients.
/lib	Symbolic link to the /usr/lib directory.
/tftpboot	Contains boot images and boot information for diskless clients.
/proc	Contains various process statistics and information on the system.

4.16 Header files

A single set of header files under /usr/include supports both 32-bit and 64-bit build environments. The directory /usr/include contains common header files. There are a set of subdirectories under /usr/include, which contain standard header files for different components of the system. Depending on the choices made during the installation, some header files may or may not be present. The main subdirectories are:

IN	Interactive library (File System, Attribute) header files used by libIN

Motif2.1	Motif 2.1 header files
Mrm	Motif Resource Manager header files
X11	X11 header files
Xm	Motif header files
aixif	AIX device driver interface header file
arpa	ARPA header files
diag	Diagnostics header files
gai	Graphic adapter interface header files
graphics	Graphics system header files
isc	ISO OSI header files
isode	ISO development environment header files
j2	JFS2 (Journal File System 2) header files
jfs	JFS (Journal File System) header files
net	Miscellaneous network header files
netinet	Internet Standard Protocol header files
netiso	OSI over TCP/IP header files
netns	XNS protocol header files
nfs	Network File System header files for backward compatibility
nsl	Network Service Library (libnsl.so) header files. Users of this library should use oncplus and tirpc header files
oncplus	Open Network Computing Plus (Network File System) header files
protocols	Berkeley service protocol header files
rpc	Client side remote procedure call header files
rpcsvc	Server side remote procedure call header files
sys	AIX system (Kernel data) header files
tirpc	Transport Independent remote procedure call header files
udi	Uniform Driver Interface header files
uil	User interface language compiler header files

4.16.1 Maximums and minimums

The POSIX and ANSI standards require that certain values are defined in /usr/include/limits.h and /usr/include/float.h. These are shown in Section 3.6.1, “C and C++ data type sizes in AIX 5L” on page 32.

Other relevant values are listed in Table 19.

Table 19. Limits imposed by AIX 5L

Description	Limit	Comment
Number of processes per user	Range: 1 to 131072 Default: 40	This is a safeguard to prevent users from creating too many processes.

Description	Limit	Comment
Size of the ARG/ENV list (in 4 KB blocks) when running exec() subroutines.	Range: 6 to 128 Default: 6	This is a mechanism to prevent the exec() subroutines from failing if the argument list is too long.
Architectural maximum file size	4 PetaBytes (JFS2)[1] 64 GB (JFS)[1]	For JFS, it is required that the file system has been enabled for large files, else the maximum file size is 2 GB.
Architectural maximum file system size	4 PetaBytes (JFS2)[1] 1 TeraByte (JFS)[1]	
Maximum file size tested	1 TeraByte (JFS2)[1] 64 GB (JFS)[1]	
Maximum file system size	1 TeraByte (JFS2)[1] 1 TeraByte (JFS)[1]	
Soft[2] file size in 512 bytes blocks	Default: 2097151	
Soft[2] core file size in 512 bytes blocks	Default: 2097151	
Soft[2] per process CPU time limit in seconds	Default: -1	Unlimited
Soft[2] data segment size in 512 bytes blocks	Default: 262144	
Soft[2] stack segment size in 512 bytes blocks	Default: 65536	
Soft[2]real memory usage in 512 bytes blocks	Default: 65536	
Soft[2] file descriptor limit	Default: 2000	

Description	Limit	Comment
[1] PetaBytes equals 1,048,576 GigaBytes, TeraBytes equals 1,024 GigaBytes, JFS2 stands for Journaled File System 2, and JFS stands for Journaled File System. On Itanium-based systems, only JSF2 is supported. [2] Soft limits can be changed using the `ulimit` command by any user, up to the maximum specified by hard limits, which can only be changed with root user authority.		

4.16.2 Limiting resource usage with WLM

The minimum and maximum use of certain system resources can be controlled using AIX Workload Manager (WLM).

WLM is included in AIX 5L and can be used to define different classes of service for jobs, as well as specify attributes for those classes. These attributes specify minimum and maximum amounts of CPU, physical memory, and disk I/O throughput to be allocated to a class.

WLM then automatically assigns jobs to classes using class assignment rules provided by a system administrator. The classification criteria are based on the value of a set of attributes of the process, such as user ID, group ID, name of the application file, type of process, and application tag.

For more details on WLM, refer to the IBM Redbook *AIX 5L Workload Manager (WLM)*, SG24-5977.

Chapter 5. Porting

This chapter describes steps you have to perform when porting code to AIX 5L. System-specific caveats are addressed and summarized tips and guidelines are presented.

5.1 Code clean - preparing your source code

For AIX 5L, the application programming interfaces (APIs) have been upgraded to allow a single set of source code to build either 32-bit or 64-bit versions of applications. This allows you to offer normal and extended capacity versions of your applications from the same source code.

Code clean is the process of updating your source code so that it:

- Uses the new multi-platform APIs
- Uses the new and revised data types
- Uses the updated function calls
- Eliminates the use of features that have been made obsolete

The code clean process has several elements:

- Selects an appropriate porting model (naming, directories, data type use, and conditional compilation macros).
- Revises your source code so that it builds on both 32-bit and 64-bit environments, including corrections for data type agreement and algorithm updates.
- Compiles updated source with both 32-bit and 64-bit compilers.
- Uses appropriate tools, such as `lint`, to warn you about possible incompatibilities.
- Performs regular regression tests that cover the complete code base in both 32-bit and 64-bit data models.

The most important feature of a code clean is its ability to add reliability, robustness, and maintainability to applications on all your supported platforms, as maintenance and support is usually rather expensive.

5.1.1 Appropriate porting model

In order to facilitate the porting process, you should:

- Choose the right programming model for you.

- Use a naming scheme for directories and file names that helps you identify easily the type of code module.
- Use the appropriate data types for the cardinality of variables.
- Use conditional compilation macros, such as __64BIT__, to separate 32-bit and 64-bit portions within your source code.

The goal is to write a *single* piece of source code that supports both 32-bit and 64-bit computing environments.

5.1.2 API revisions

The updated operating system APIs have preserved the names and semantics of many of the familiar data types and have revised the definition of others to scale with the capacity of the target architecture (for example long). In addition, sized data types are available (usually as typedefs), allowing exact specification of the precision of any data value in your source, for example int16_t and int32_t.

This combination of types and typedefs allows revision of data types used in source code to offer the desired features and capacity on either platform.

Certain features have also been deprecated, or scheduled for obsolescence. These features are being made obsolete because they could not represent the larger capacities of LP64, or could not accept or supply 64-bit sized quantities on 64-bit platforms.

The following API revisions comprise the majority of code clean:

- Update code that mixes pointers and integers: Change any (int)pointer casts to (uintptr_t)pointer
- Fine-tune variable and structure field sizes: Do not waste space using a long if an int will do
- Optimize field ordering in structures: Minimize padding by placing scalable types early in the struct definition

5.1.3 Data type agreement

If source code must be built for both 32-bit and 64-bit environments, you must avoid use of the long data type. You can avoid this in either of two ways:

- By using ANSI C data types that scale properly (like intptr_t and uintptr_t)
- By using a data type like int32long64_t and uint32long64_t, which is a conditionally compiled typedef

A number of derived data types have changed to now represent 64-bit quantities in the 64-bit application environment (see Section 5.2, “System derived data types” on page 120). This change does not affect 32-bit applications. However, any 64-bit application that consumes or exports data described by these types needs to be re-evaluated for correctness.

5.1.4 Algorithm updates

Many algorithms need to be modified for 64-bit environments, as they may assume certain data type sizes and may optimize for those sizes. Also, some algorithms make use of and presume particular bits or bit patterns in data or pointers. If the size of data or pointers changes, you may have to change these algorithms. These modifications can include:

- Hashing algorithms often use pointers or seek keys as the hash keys that will yield a good distribution only when there is a lot of significant data to scramble. In 32-bit addressing, there may be 30 out of 32 bits varying, thus, a fair distribution. With the same set of data in 64-bit addressing, however, there may be only 35 significant bits varying out of 64 bits, worsening the fair distribution of the hashing algorithm. Therefore, you have to update algorithms that hash 64-bit pointers or integers to adequately "randomize" those bits actually used.
- Smarter heap allocation routines that place records of pointer chains close together and “data structure walking” to take advantage of the above.
- Correction of any mixing of double and long data types, as they can lose bits of precision.
- Paying attention to possible differences in shifting and bit masking using hexadecimal constants.

After your code has been made 64-bit safe, you should review it again to verify that all algorithms and data structures still make sense. Some of the data types are larger in 64-bit environments, so data structures might take up more space, which might influence the performance of your code as well. Given these concerns, you may need to modify your code accordingly.

5.1.5 Software correctness

Correcting the type scalability (size polymorphism) is only part of the code clean process. Unfortunately, it is not possible to automate the whole process. Only a human can decide when algorithms must be updated and therefore avoid undesirable side effects like:

- Increase in data structure sizes and alignment
- Changed function return values of system derived types

5.2 System derived data types

The different data types available on a system can be divided into two different classes:

Base data types	Base data types, also known as primitive types or built-in types, are all data types already defined by the C language specification. They can be used without having to be defined with `typedef`. However, the characteristics of some base types are not necessarily defined and are specific to each implementation and hardware platform. For example, the size and alignment of the int and long data types are hardware-dependent characteristics.
Derived data types	A derived data type is one that is defined with `typedef` as a derivative or structure of existing base types or other derived types. For example, size_t and FILE are derived types defined in the header file <stdio.h>.

Using the system derived data types helps make code 32-bit and 64-bit safe, since the derived data types themselves must be safe for both the ILP32 and LP64 data models. In general, using derived data types to allow for change is a good programming practice. Should the data model change in the future, or when porting to a different platform, only the system derived data types need to change rather than the application.

Most kernel interfaces have defined their own derived data types. For example, the data type of a UNIX process ID is defined as pid_t instead of int or long. These interface-specific data types can be found in their corresponding header files. Other general derived data types can be found in the header files <sys/types.h> and <inttypes.h>.

5.2.1 Data types defined by <sys/types.h>

Table 20 lists some commonly used system-derived types defined on AIX 5L along with their sizes in the two supported programming models (ILP32 and LP64).

Table 20. System derived type description

Derived type	**Size**		**Description**
	ILP32	**LP64**	
cptr32, __cptr32	32	32	Fixed size 32-bit char* pointer

Derived type	Size		Description
	ILP32	LP64	
cptr64, __cptr64	64	64	Fixed size 64-bit char* pointer
ptr32, __ptr32	32	32	Fixed size 32-bit generic pointer
ptr64, __ptr64	64	64	Fixed size 64-bit generic pointer
caddr_t	32	64	Memory address
clock_t	32	32	Represents the system time in clock ticks
dev_t	32	64	Used for device numbers
fpos_t	32/64[1]	64	Used for file offsets
gid_t	32	32	Group ID
mode_t	32	32	File mode
off_t	32/64[1]	64	Used for file sizes and offsets
pid_t	32	64	Process ID within 32-bit or 64-bit kernel
ptrdiff_t	32	64	Signed integral type for result of pointer subtraction
size_t	32	64	Size of objects in memory in bytes
ssize_t	32	64	Used to return a count of bytes or an error indication
time_t	32	64	Used for time in seconds since 01.01.1970
wint_t	32	32	Wide characters
1 Size is 64 bits when _LARGE_FILES defined			

Table 21 lists those derived types along with their actual base types in ILP32 and LP64 mode.

Table 21. Relation between system derived and base data types

Derived type	Definition	
	ILP32	LP64
cptr32, __cptr32	char *	unsigned int
cptr64, __cptr64	unsigned long long	char *
ptr32, __ptr32	void *	unsigned int

Derived type	Definition	
	ILP32	LP64
ptr64, __ptr64	unsigned long long	void *
caddr_t	char *	char *
clock_t	int	int
dev_t	unsigned_int	unsigned long
fpos_t	long[1]	long[1]
gid_t	unsigned int	unsigned int
mode_t	unsigned int	unsigned int
off_t	long[1]	long[1]
pid_t	int	long
ptrdiff_t	long	long
size_t	unsigned long	unsigned long
ssize_t	long	long
time_t	int	long
wint_t	int	int
[1] Type is long long when _LARGE_FILES defined		

5.2.2 Data types defined by <inttypes.h>

The system header file <inttypes.h> provides system derived data types that help programmers make their code compatible with explicitly sized data items (8-bit, 16-bit, 32-bit, and 64-bit objects), independent of the compilation environment. The file is part of an ANSI C proposal and tracks the ISO/JTC1/SC22/WG14 C committee's working draft for the revision of the current ISO C standard, ISO/IEC 9899:1990 Programming language - C.

The basic features regarding data types provided by <inttypes.h> are:

- Fixed size integer data types
- Other helpful data types

5.2.2.1 Fixed size integer data types

The fixed size integer data types provided by <inttypes.h> include both signed and unsigned integer data types, as shown in Table 22.

Table 22. Fixed size integer data types defined by <inttypes.h>

Fixed size data type		ILP32 mode		LP64 mode	
Signed	**Unsigned**	**Size**	**Alignment**	**Size**	**Alignment**
int8_t	uint8_t	1	1	1	1
int16_t	uint16_t	2	2	2	2
int32_t	uint32_t	4	4	4	4
int64_t	uint64_t	8	4	8	8

Derived data types defined as the smallest signed integer data types that can hold the specified number of bits are shown in Table 23.

Table 23. Derived data types holding the smallest signed integer data types

Signed data type	ILP32 mode			LP64 mode		
	Size	**Align**	**Data type**	**Size**	**Align**	**Data type**
int_least8_t	1	1	signed char	1	1	signed char
int_least16_t	2	2	signed short	2	2	signed short
int_least32_t	4	4	signed int	4	4	signed int
int_least64_t	8	4	signed long long	8	8	signed long

Derived data types defined as the smallest unsigned integer data types that can hold the specified number of bits are shown in Table 24.

Table 24. Derived data types holding the smallest unsigned integer data types

Unsigned data type	ILP32 mode			LP64 mode		
	Size	**Align**	**Data type**	**Size**	**Align**	**Data type**
uint_least8_t	1	1	unsigned char	1	1	unsigned char
uint_least16_t	2	2	unsigned short	2	2	unsigned short
uint_least32_t	4	4	unsigned int	4	4	unsigned int
uint_least64_t	8	4	unsigned long long	8	8	unsigned long

Derived data types defined as the most efficient signed integer data types that can hold the specified number of bits are shown in Table 25.

Table 25. Most efficient signed data types with the specified number of bits

Signed data type	ILP32 mode			LP64 mode		
	Size	Align	Data type	Size	Align	Data type
int_fast8_t	1	1	signed char	1	1	signed char
int_fast16_t	2	2	int32_t	2	2	int32_t
int_fast32_t	4	4	int32_t	4	4	int32_t
int_fast64_t	8	4	int64_t	8	8	int64_t

Derived data types defined as the most efficient unsigned integer data types that can hold the specified number of bits are shown in Table 26.

Table 26. Most efficient unsigned data types with the specified number of bits

Unsigned data type	ILP32 mode			LP64 mode		
	Size	Align	Data type	Size	Align	Data type
uint_fast8_t	1	1	unsigned char	1	1	unsigned char
uint_fast16_t	2	2	uint32_t	2	2	uint32_t
uint_fast32_t	4	4	uint32_t	4	4	uint32_t
uint_fast64_t	8	4	uint64_t	8	8	uint64_t

Note

These fixed size data types should not be used indiscriminately. For example, you can still use int for such things as loop counters, and long can still be used for array indexes. On the other hand, use fixed size data types for explicit binary representations of, for example,:

- Fixed size binary disk data
- Fixed size network data
- Hardware registers
- Binary interface specifications
- Binary data structures

5.2.2.2 Other helpful data types

Other useful types provided by <inttypes.h> include signed and unsigned integer data types (intptr_t and uintptr_t) large enough to hold any data pointer; that is, data pointers can be assigned into or from these integer data types without losing precision. These derived pointer data types are shown in Table 27.

Table 27. Derived integer data types to hold any data pointer

Derived data type	Integral data type	ILP32 mode		LP64 mode	
		Size	Align	Size	Align
intptr_t	signed long	4	4	8	8
uintptr_t	unsigned long	4	4	8	8

Using the uintptr_t type as the integral type for pointers is a better option than using a fundamental type, such as unsigned long. Even though an unsigned long is the same size as a pointer in both the ILP32 and LP64 data models, the use of the uintptr_t requires only the definition of uintptr_t to change when a different data model is used. This makes the code more portable to other systems and it is also a clearer way to express your intentions in C.

The intptr_t and uintptr_t types are extremely useful for casting pointers when performing address arithmetic. They should be used instead of long or unsigned long for this purpose.

Other useful types provided by <inttypes.h> include signed and unsigned integer data types (intmax_t and uintmax_t) which are defined to be the longest (in bits) signed and unsigned integer data types available. These derived integer data types are shown in Table 28.

Table 28. Derived integer data types to hold maximum integer values

Derived data type	ILP32 mode			LP64 mode		
	Defined as	Size	Align	Defined as	Size	Align
intmax_t	int32_t	4	4	int64_t	8	8
uintmax_t	uint32_t	4	4	uint64_t	8	8

5.3 System derived constants and macros

Using the system derived constants and macros helps make code 32-bit and 64-bit safe, since the derived constants and macros themselves must be safe for both the ILP32 and LP64 data models. In general, using derived constants and macros to allow for change is a good programming practice. Should the data model change in the future, or when porting to a different platform, only the system derived constants and macros need to change rather than the application.

System derived constants and macros can be found in the system header files <limits.h> and <inttypes.h>.

5.3.1 Constants and macros defined by <limits.h>

The system header file <limits.h> defines constants that should be used (instead of the literal values) to ensure greater portability.

Table 29 shows integer constants with the same value in ILP32 mode and LP64 mode.

Table 29. Integer constants defined by <limits.h> for ILP32 and LP64 mode

Constant	Description	Numeric value
CHAR_BIT	Number of bits per char	8
WORD_BIT	Number of bits per int	32
CHAR_MAX	Biggest char	127[1] or UCHAR_MAX[2]
CHAR_MIN	Smallest char	-128[1] or 0[2]
SCHAR_MAX	Biggest signed char	127
SCHAR_MIN	Smallest signed char	-SCHAR_MAX-1
UCHAR_MAX	Biggest unsigned char	255
SHRT_MAX	Biggest signed short	32,767
SHRT_MIN	Smallest signed short	-SHRT_MAX-1
USHRT_MAX	Biggest unsigned short	65,535
INT_MAX	Biggest signed int	2,147,483,647
INT_MIN	Smallest signed int	-INT_MAX-1
UINT_MAX	Biggest unsigned int	4,294,967,295
[1] __ia64 defined and default char type is signed [2] default char type is unsigned		

Table 30 shows integer constants for ILP32 mode that have a different value in LP64 mode due to different data type sizes in both programming models.

Table 30. Integer constants defined by <limits.h> for ILP32 mode

Constant	Description	Numeric value
LONG_BIT	Number of bits per long	32
LONG_MAX	Biggest signed long	INT_MAX
LONG_MIN	Smallest signed long	-INT_MIN
ULONG_MAX	Biggest unsigned long	UINT_MAX

Table 31 shows integer constants for LP64 mode that have a different value in ILP32 mode due to different data type sizes in both programming models.

Table 31. Integer constants defined by <limits.h> for LP64 mode

Constant	Description	Numeric value
LONG_BIT	Number of bits per long	64
LONG_MAX	Biggest signed long	9,223,372,036,854,775,807
LONG_MIN	Smallest signed long	-LONG_MAX-1
ULONG_ MAX	Biggest unsigned long	18,446,744,073,709,551,615

5.3.2 Constants and macros defined by <inttypes.h>

The system header file <inttypes.h> provides constants that help programmers make their code compatible with explicitly sized data items (8-bit, 16-bit, 32-bit, and 64-bit objects), independent of the compilation environment. The file is part of an ANSI C proposal and tracks the ISO/JTC1/SC22/WG14 C committee's working draft for the revision of the current ISO C standard, ISO/IEC 9899:1990 Programming language - C.

The basic features with respect to constants and macros provided by <inttypes.h> are:

- Limit constants
- Format string macros

5.3.2.1 Limit constants

The implementation limits defined by <inttypes.h> are constants specifying the minimum and maximum values of various integer data types. This includes minimum and maximum values of each of the fixed size data types, as shown in Table 32.

Table 32. Constants for the minimum and maximum of some integer types

Constant	Description	Numeric value
INT8_MIN	Minimum signed 8-bit value	-128
INT8_MAX	Maximum signed 8-bit value	127
UINT8_MAX	Maximum unsigned 8-bit value	255
INT16_MIN	Minimum signed 16-bit value	-32,768
INT16_MAX	Maximum signed 16-bit value	32,767
UINT16_ MAX	Maximum unsigned 16-bit value	65,536
INT32_MIN	Minimum signed 32-bit value	-2,147,483,648
INT32_MAX	Maximum signed 32-bit value	2,147,483,647
UINT32_ MAX	Maximum unsigned 32-bit value	4,294,967,295
INT64_MIN	Minimum signed 64-bit value	-9,223,372,036,854,775,808
INT64_MAX	Maximum signed 64-bit value	9,223,372,036,854,775,807
UINT64_ MAX	Maximum unsigned 64-bit value	18,446,744,073,709,551,615

The minimum and maximum for each of the least sized integer data types are also defined by <inttypes.h>. They are shown in Table 33.

Table 33. Constants for the minimum and maximum of least sized integer types

Constant	Description	Numeric value
INT_LEAST8_MIN	Minimum signed least 8-bit value	INT8_MIN
INT_LEAST16_MIN	Minimum signed least 16-bit value	INT16_MIN
INT_LEAST32_MIN	Minimum signed least 32-bit value	INT32_MIN
INT_LEAST64_MIN	Minimum signed least 64-bit value	INT64_MIN
INT_LEAST8_MAX	Maximum signed least 8-bit value	INT8_MAX
INT_LEAST16_MAX	Maximum signed least 16-bit value	INT16_MAX
INT_LEAST32_MAX	Maximum signed least 32-bit value	INT32_MAX

Constant	Description	Numeric value
INT_LEAST64_MAX	Maximum signed least 64-bit value	INT64_MAX
UINT_LEAST8_MIN	Minimum unsigned least 8-bit value	UINT8_MIN
UINT_LEAST16_ MIN	Minimum unsigned least 16-bit value	UINT16_MIN
UINT_LEAST32_ MIN	Minimum unsigned least 32-bit value	UINT32_MIN
UINT_LEAST64_ MIN	Minimum unsigned least 64-bit value	UINT64_MIN
UINT_LEAST8_MAX	Maximum signed least 8-bit value	UINT8_MAX
UINT_LEAST16_ MAX	Maximum signed least 16-bit value	UINT16_MAX
UINT_LEAST32_ MAX	Maximum signed least 32-bit value	UINT32_MAX
UINT_LEAST64_ MAX	Maximum signed least 64-bit value	UINT64_MAX

The minimum and maximum values for each of the most efficient integer data types are also defined by <inttypes.h>. They are shown in Table 34.

Table 34. Minimum and maximum constants for the most efficient integer types

Constant	Description	Numeric value
INT_FAST8_MIN	Minimum signed 8-bit value	INT8_MIN
INT_FAST16_MIN	Minimum signed 16-bit value	INT32_MIN
INT_FAST32_MIN	Minimum signed 32-bit value	INT32_MIN
INT_FAST64_MIN	Minimum signed 64-bit value	INT64_MIN
INT_FAST8_MAX	Maximum signed 8-bit value	INT8_MAX
INT_FAST16_MAX	Maximum signed 16-bit value	INT32_MAX
INT_FAST32_MAX	Maximum signed 32-bit value	INT32_MAX
INT_FAST64_MAX	Maximum signed 64-bit value	INT64_MAX
UINT_FAST8_MIN	Minimum unsigned 8-bit value	UINT8_MIN
UINT_FAST16_MIN	Minimum unsigned 16-bit value	UINT32_MIN
UINT_FAST32_MIN	Minimum unsigned 32-bit value	UINT32_MIN

Constant	Description	Numeric value
UINT_FAST64_MIN	Minimum unsigned 64-bit value	UINT64_MIN
UINT_FAST8_MAX	Maximum signed 8-bit value	UINT8_MAX
UINT_FAST16_MAX	Maximum signed 16-bit value	UINT32_MAX
UINT_FAST32_MAX	Maximum signed 32-bit value	UINT32_MAX
UINT_FAST64_MAX	Maximum signed 64-bit value	UINT64_MAX
INTFAST_MIN	Minimum signed integer value	INT32_MIN
INTFAST_MAX	Maximum signed integer value	INT32_MAX
UINTFAST_MAX	Maximum unsigned integer value	UINT32_MAX

The minimum and maximum value of the largest supported integer data types are also defined in <inttypes.h> and shown in Table 35.

Table 35. Minimum and maximum constants for the largest integer types

Constant	Description	ILP32 mode	LP64 mode
INTMAX_MIN	Minimum largest signed integer value	INT32_MIN	INT64_MIN
INTMAX_MAX	Maximum largest signed integer value	INT32_MAX	INT64_MAX
UINTMAX_MAX	Maximum largest unsigned integer value	UINT32_MAX	UINT64_MAX

The maximum value of the largest integer data type which can fully contain any pointer data type without precision loss is also defined in <inttypes.h>, as shown in Table 36.

Table 36. Maximum constants for the largest pointer data types

Constant	Description	ILP32 mode	LP64 mode
INTPTR_MAX	Maximum signed integer value	INT32_MAX	INT64_MAX
UINTPTR_MAX	Maximum unsigned integer value	UINT32_MAX	UINT64_MAX

5.3.2.2 Format string macros

The <inttypes.h> header file defines a number of format string macros for use with the different versions of the printf and scanf family of routines.

(s)printf() subroutine

Format string macros for the (s)printf() subroutines for different conversion specifiers are also defined in <inttypes.h>. See Table 37 for more information.

Table 37. Predefined format string macros for (s)printf

Format macro	Description	ILP32	LP64
PRId8	signed 8-bit value	%hd	%hd
PRId16	signed 16-bit value	%hd	%hd
PRId32	signed 32-bit value	%d	%d
PRId64	signed 64-bit value	%lld	%ld
PRIdLEAST8	signed least 8-bit value	%hd	%hd
PRIdLEAST16	signed least 16-bit value	%hd	%hd
PRIdLEAST32	signed least 32-bit value	%d	%d
PRIdLEAST64	signed least 64-bit value	%lld	%ld
PRIdFAST8	signed most efficient 8-bit value	%hd	%hd
PRIdFAST16	signed most efficient 16-bit value	%hd	%hd
PRIdFAST32	signed most efficient 32-bit value	%d	%d
PRIdFAST64	signed most efficient 64-bit value	%lld	%ld

Additional predefined format string macros for the conversion specifiers i, u, o, x, and X are also available. They can be derived according to Table 38.

Table 38. Rules to derive other format string macros from signed format

Conversion specifier	Interpretation
signed (second variant)	Replace d with i
unsigned	Replace d with u
octal	Replace d with o
hexadecimal (small letters)	Replace d with x
hexadecimal (capital letters)	Replace d with X

For example, to obtain the unsigned 32-bit value format string macro from the signed one, only the d in PRId32 has to be changed to an u, thus PRIu32.

In addition, there are also format string macros defined for the maximum size data types, the most efficient data types, and pointers. They are shown in Table 39.

Table 39. Format string macros

Format macro	Description	ILP32	LP64
PRIdMAX	Maximum integer decimal	%lld	%ld
PRIoMAX	Maximum integer octal	%llo	%lo
PRIxMAX	Maximum integer hexadecimal	%llx	%lx
PRIuMAX	Maximum integer unsigned	%llu	%lu
PRIdFAST	Most efficient integer decimal	%d	%d
PRIoFAST	Most efficient integer octal	%o	%o
PRIxFAST	Most efficient integer hexadecimal	%x	%x
PRIuFAST	Most efficient integer unsigned	%u	%x
PRIdPTR	Pointer in decimal form	%ld	%ld
PRIoPTR	Pointer in octal form	%lo	%lo
PRIxPTR	Pointer in hexadecimal form	%lx	%lx
PRIXPTR	Pointer in hexadecimal form	%lX	%lX
PRIuPTR	Pointer in unsigned decimal form	%lu	%lu

(s)scanf() subroutine

Format string macros for the (s)scanf() subroutine for different conversion specifiers are also defined in <inttypes.h>, as shown in Table 40.

Table 40. Predefined format string macros for (s)scanf()

Constant	Description	ILP32	LP64
SCNd16	signed 16-bit value	%hd	%hd
SCNd32	signed 32-bit value	%d	%d
SCNd64	signed 64-bit value	%lld	%ld

Additional predefined format string macros for the conversion specifiers i, u, o, and x are also available. They can also be derived according to Table 38 on page 131. For example, to obtain the unsigned 32-bit value format string macro, only the ‘d’ in SCNd32 has to be changed to an ‘u’, thus SCNud32.

In addition, there are also format string macros defined for the maximum size data types, the most efficient data types, and pointers. They are shown in Table 41.

Table 41. Format string macros

Format macro	Description	ILP32	LP64
SCNdMAX	Maximum integer decimal	%lld	%ld
SCNoMAX	Maximum integer octal	%llo	%lo
SCNxMAX	Maximum integer hexadecimal	%llx	%lx
SCNuMAX	Maximum integer unsigned	%llu	%lu
SCNdFAST	Most efficient integer decimal	%d	%d
SCNoFAST	Most efficient integer octal	%o	%o
SCNxFAST	Most efficient integer hexadecimal	%x	%x
SCNuFAST	Most efficient integer unsigned	%u	%x
SCNdPTR	Pointer in decimal form	%ld	%ld
SCNoPTR	Pointer in octal form	%lo	%lo
SCNxPTR	Pointer in hexadecimal form	%lx	%lx
SCNuPTR	Pointer in unsigned decimal form	%lu	%lu

5.4 System specific differences

In this section, we take a closer look at system specific differences between the different operating systems and AIX 5L in terms of:

- System derived data types
- Application Programming Interfaces
- Miscellaneous

It is very important to know the exact size of employed data types throughout your program and the differences with regards to system calls.

5.4.1 System derived data types

Depending on the employed data model (32-bit or 64-bit), system derived data types can have different byte sizes. The following subsections compare the most common system derived data types and their actual implementation for the different platforms.

5.4.1.1 Pointer data types cptr32, __cptr32, cptr64, and __cptr64

The purpose of the pointer data types cptr32, __cptr32, cptr64, and __cptr64 is meant to be fixed size pointers to a char. See Table 42 for more information.

Table 42. Pointer data types cptr32, __cptr32, cptr64, and __cptr64

OS	cptr32, __cptr32	cptr64, __cptr64
AIX 4.x 32-bit	char *	unsigned int
AIX 4.3 64-bit	unsigned int	char *
AIX 5L ILP32	char *	unsigned long long
AIX 5L LP64	unsigned int	char *
Solaris 2.6	N/A	N/A
Solaris 7	N/A	N/A
Solaris 8	N/A	N/A
HP-UX 10.xx	N/A	N/A
HP-UX 11.xx	N/A	N/A
Compaq Tru64	N/A	N/A
SGI IRIX 6.5	N/A	N/A

5.4.1.2 Pointer data types ptr32, __ptr32, ptr64, and __ptr64

The purpose of the pointer data types ptr32, __ptr32, ptr64, and __ptr64 is meant to be fixed size generic pointers. See Table 43 for more information.

Table 43. Pointer data types ptr32, __ptr32, ptr64, and __ptr64

OS	ptr32, __ptr32	ptr64, __ptr64
AIX 4.x 32-bit	void *	unsigned int
AIX 4.3 64-bit	unsigned int	void *
AIX 5L ILP32	void *	unsigned long long
AIX 5L LP64	unsigned int	void *
Solaris 2.6	N/A	N/A
Solaris 7	N/A	N/A
Solaris 8	N/A	N/A
HP-UX 10.xx	N/A	N/A

OS	ptr32, __ptr32	ptr64, __ptr64
HP-UX 11.xx	N/A[1]	N/A
Compaq Tru64	N/A	N/A
SGI IRIX 6.5	N/A	N/A
[1]Type ptr32_t is defined as an unsigned 32-bit integer.		

5.4.1.3 Pointer data types caddr_t, intptr_t, uintptr_t, and ptrdiff_t

The pointer data type caddr_t should be used for “core” memory access, while the pointer data types intptr_t and uintptr_t are signed and unsigned integer data types large enough to hold any pointer value without loss of precision. The pointer data type ptrdiff_t is a signed integer data type that should be used for the result of a pointer subtraction. See Table 44 for more information.

Table 44. Pointer data types caddr_t, intptr_t, uintptr_t, and ptrdiff_t

OS	caddr_t	intptr_t	uintptr_t	ptrdiff_t
AIX 4.x 32-bit	char *	long	unsigned long	long
AIX 4.3 64-bit	char *	long	unsigned long	long
AIX 5L ILP32	char *	long	unsigned long	long
AIX 5L LP64	char *	long	unsigned long	long
Solaris 2.6 ILP32	char *	int	unsigned int	int
Solaris 2.6 LP64	char *	long	unsigned long	int
Solaris 7 ILP32	char *	int	unsigned int	int
Solaris 7 LP64	char *	long	unsigned long	long
Solaris 8 ILP32	char *	int	unsigned int	int
Solaris 8 LP64	char *	long	unsigned long	long
HP-UX 10.xx	char *	long	unsigned long	int
HP-UX 11.xx	char *	long	unsigned long	long
Compaq Tru64	char *	long	unsigned long	long
SGI IRIX 6.5 ILP32	char *	long	unsigned long	long
SGI IRIX 6.5 LP64	char *	long	unsigned long	long

5.4.1.4 Data types clock_t, dev_t, and time_t

The data type clock_t should be used for representations of the system in the form of clock ticks, while the data type time_t should be used for representations of time in the form of the number of seconds that have passed since January 1st, 1970. The data type dev_t should be used for device numbers. See Table 45 for more information.

Table 45. Data types clock_t, dev_t, and time_t

OS	clock_t	dev_t	time_t
AIX 4.x 32-bit	int	unsigned int	int
AIX 4.3 64-bit	int	unsigned long	int
AIX 5L ILP32	int	unsigned int	int
AIX 5L LP64	int	unsigned long	long
Solaris 2.6	long	unsigned long	long
Solaris 7	long	unsigned long	long
Solaris 8	long	unsigned long	long
HP-UX 10.xx	unsigned int	int	long
HP-UX 11.xx	unsigned int	int	long
Compaq Tru64	int	int	int[1]
SGI IRIX 6.5 ILP32	long	unsigned long	long
SGI IRIX 6.5 LP64	int	unsigned int	int
[1]The type time64_t is defined as long when _TIME64_T is defined			

5.4.1.5 Data types fpos_t, fpos64_t, off_t, and off64_t

The data types fpos_t and fpos64_t are used to represent file offsets for the positioning of file pointers, with fpos64_t being used for file offsets with large file support (file sizes greater than 2 GB).

The data types off_t and off64_t are used to represent file sizes and offsets for the positioning of file pointers, with off64_t being used in conjunction with with large file support (file sizes greater than 2 GB). See Table 46 for more information.

Table 46. Data types fpos_t, fpos64_t, off_t, and off64_t

OS	fpos_t	fpos64_t	off_t	off64_t
AIX 4.x 32-bit	long[1]	long long	long	long long

OS	fpos_t	fpos64_t	off_t	off64_t
AIX 4.3 64-bit	long[1]	long long	long	long long
AIX 5L ILP32	long[1]	long long[5]	long[1]	long long
AIX 5L LP64	long[1]	long long[5]	long[1]	long long
Solaris 2.6	long[3]	long long	long[3]	long long
Solaris 7 ILP32	long[3]	long long	long[3]	long long
Solaris 7 LP64	long	long	long	long
Solaris 8 ILP32	long[3]	long long	long[3]	long long
Solaris 8 LP64	long	long	long	long
HP-UX 10.xx	int[3]	long long	int[4]	long long
HP-UX 11.xx ILP32	int	long long	long[3]	long long
HP-UX 11.xx LP64	long	long	long	long
Compaq Tru64	long	N/A	long[6]	N/A
SGI IRIX 6.5 ILP32	long long	long long	long long	long long
SGI IRIX 6.5 LP64	long	long	long	long

[1] Type is long long when _LARGE_FILES defined.
[2] Type is 64-bit with large file support.
[3] Type is long long with large file support.
[4] Type is long long when _FILE64 defined.
[5] Type is long long when _LARGE_FILES_API defined.
[6] Type is unsigned long when _KERNEL defined.

5.4.1.6 Data types gid_t, mode_t, pid_t, and uid_t

The data type gid_t is used to represent the group ID of a user, while the data type uid_t is used for the user ID. The data type mode_t is used to indicate the mode of a file, while the data type pid_t is used to identify the different processes with unique numbers. See Table 47 for more information.

Table 47. Data types gid_t, mode_t, pid_t, and uid_t

OS	gid_t	mode_t	pid_t	uid_t
AIX 4.x 32-bit	unsigned int	unsigned int	int	unsigned int
AIX 4.3 64-bit	unsigned int	unsigned int	long	unsigned int
AIX 5L ILP32	unsigned int	unsigned int	int	unsigned int

OS	gid_t	mode_t	pid_t	uid_t
AIX 5L LP64	unsigned int	unsigned int	long	unsigned int
Solaris 2.6	long	unsigned long	long	long
Solaris 7 ILP32	long	unsigned long	long	long
Solaris 7 LP64	int	unsigned int	int	int
Solaris 8 ILP32	long	unsigned long	long	long
Solaris 8 LP64	int	unsigned int	int	int
HP-UX 10.xx	int	unsigned short	int	int
HP-UX 11.xx	int	unsigned short	int	int
Compaq Tru64	unsigned int	unsigned int	int	unsigned int[1]
SGI IRIX 6.5 ILP32	long	unsigned long	long	long
SGI IRIX 6.5 LP64	int	unsigned int	int	int
[1] The type uid_t is defined as an int when _XOPEN_SOURCE_EXTENDED is defined.				

5.4.1.7 size_t, ssize_t, and wint_t

The data types size_t and ssize_t should be used in conjunction with the size of objects in memory and to return a count of bytes or an error indication, respectively. The data type wint_t is required to represent the wide character code value as well as the end-of-file (EOF) marker, when programming with multibyte and wide character subroutines. See Table 48 for more information.

Table 48. Data types size_t, ssize_t, and wint_t

OS	size_t	ssize_t	wint_t
AIX 4.x 32-bit	unsigned long	long	int
AIX 4.3 64 bit	unsigned long	long	int
AIX 5L ILP32	unsigned long	long	int
AIX 5L LP64	unsigned long	long	int
Solaris 2.6	unsigned int	int	long
Solaris 7 ILP32	unsigned int	int	long
Solaris 7 LP64	unsigned long	long	int
Solaris 8 ILP32	unsigned int	int	long
Solaris 8 LP64	unsigned long	long	int

OS	size_t	ssize_t	wint_t
HP-UX 10.xx ILP32	unsigned long	long	unsigned int
HP-UX 10.xx LP64	unsigned long long	long long	unsigned int
HP-UX 11.xx	unsigned long	long	unsigned int
Compaq Tru64	unsigned int	long	unsigned int
SGI IRIX 6.5 ILP32	unsigned int	int	long
SGI IRIX 6.5 LP64	unsigned long	long	int

5.4.2 Application Programming Interfaces

The 32-bit Application Programming Interfaces (APIs) of AIX 5L supported in the 64-bit operating environment are the same as the APIs supported in the 32-bit operating environment. Thus, no changes are required for 32-bit applications between the 32-bit and 64-bit environments. However, recompiling as a 64-bit application can require cleanup of your code.

5.4.2.1 Memory allocation routines

The malloc() subroutine returns a pointer to a block of memory of at least the number of bytes specified by the size parameter. The block is aligned so that it can be used for any type of data.

The realloc() subroutine changes the size of the block of memory pointed to by the pointer parameter to the number of bytes specified by the size parameter and returns a new pointer to the block. The pointer specified by the pointer parameter must have been created with the malloc(), calloc(), or realloc() subroutines, and not been deallocated with the free() or realloc() subroutines. Undefined results occur if the pointer parameter is not a valid pointer.

The calloc() subroutine allocates space for an array with the specified number of elements, the specified size in bytes for each element, and initializes the allocated space to zeros. The order and contiguity of storage allocated by successive calls to the calloc() subroutine is unspecified. The pointer returned points to the first (lowest) byte address of the allocated space.

The alloca() subroutine allocates the number of bytes of space specified by the size parameter in the stack frame of the caller. This space is automatically freed when the subroutine that called the alloca() subroutine returns to its caller.

The valloc() subroutine has the same effect as malloc(), except that the allocated memory is aligned to a multiple of the value returned by sysconf(_SC_PAGESIZE). See Table 49 for more information.

Table 49. Argument type for memory allocation routines

OS	malloc	realloc	calloc	alloca	valloc
AIX 4.x	size_t	size_t	size_t	int	size_t
AIX 5L	size_t	size_t	size_t	int	size_t
Solaris 8	size_t	size_t	size_t	size_t	size_t
HP-UX 10.xx	size_t	size_t	size_t	size_t	size_t
HP-UX 11.xx	size_t	size_t	size_t	size_t	size_t
Compaq Tru64	size_t	size_t	size_t	int	size_t

As the ANSI/ISO C standard dictates, the behavior of a call to malloc() with a size argument of 0 is implementation dependent; therefore, you should guard your code against this potential problem, as malloc() can either return a NULL pointer or a valid pointer that points to a block of memory with a size of 0 bytes.

5.4.2.2 File positioning routines

The lseek() system call is used for setting the current position in a file. To indicate the displacement a data type, off_t should be passed to lseek() to indicate the wanted file offset. The llseek() system call (not available on all platforms) and lseek64() system call take an argument of type offset_t and off64_t, respectively. Both system calls are used for file pointer positioning in large files (file sizes greater than 2 GB).

When setting or getting the file positions for a file with the ANSI C functions fsetpos() or fgetpos(), respectively, the argument to specify the file position is defined to be of data type fpos_t. The system calls fsetpos64() and fgetpos64() are used for large files (file sizes greater than 2 GB) and take an argument of data type off64_t.

Table 50 summarizes the details of the file positioning routines.

Table 50. Argument types for file positioning routines

OS	lseek	llseek	lseek64	fsetpos fgetpos	fsetpos64 fgetpos64
AIX 4.x	off_t	offset_t	off64_t	fpos_t	fpos64_t
AIX 5L	off_t	offset_t	off64_t	fpos_t	fpos64_t

OS	lseek	llseek	lseek64	fsetpos fgetpos	fsetpos64 fgetpos64
Solaris 7	off_t	offset_t	off64_t	fpos_t	fpos64_t
HP-UX 10.xx	off_t	N/A	off64_t	fpos_t	fpos64_t
HP-UX 11.xx	off_t	N/A	off64_t	fpos_t	fpos64_t
Compaq Tru64	off_t	N/A	N/A	fpos_t	N/A

5.4.3 Threads

Threads are being used in more applications, and most platforms provide some form of threads support. AIX 5L is no exception and supports the POSIX threads definition. Even within the standard, there may be implementation specific operating system differences that could cause problems. Chapter 10, “POSIX threads” on page 307 describes threads and provides a good source of information to ensure correct configuration and operation.

5.4.4 The sizeof() operator

The result of the sizeof() operator is the data type size_t, which depends on the specific operating environment (32-bit or 64-bit) and employed programming model; see Table 48 on page 138 for a listing.

5.4.5 Self-modifying code

Self-modifying code is a programming technique that enables a program module to change functionality by modifying the in-memory machine instructions of the module itself. This technique is not supported on AIX 5L on either Power or Itanium-based systems. All self-modifying code must be rewritten in standard fashion, in other words, using if-clauses to determine the appropriate functionality.

5.4.6 System specific commands

Shell scripts may also need to be ported, particularly if they attempt to parse the output of system commands.

5.4.6.1 Shell scripts

Shell scripts that adhere to the POSIX standard and do not rely on system specific commands should be rather easy to adapt for AIX 5L. For security reasons, AIX 5L does not allow shell scripts to be setuid as this poses a serious security risk due to the fact that this would allow for an easy access to root user rights for any user.

5.4.6.2 The Korn shell

In AIX 5L the default shell is /usr/bin/ksh which is hardlinked to /usr/bin/psh, /usr/bin/sh, and /usr/bin/tsh. This is an implementation of the 1988 version of the Korn Shell, enhanced to be POSIX compliant.

In addition to this shell, an unmodified version of the 1993 version of ksh is supplied as /usr/bin/ksh93. This version is also POSIX compliant. With the exception of POSIX specific items, the 1993 version should be backward compatible with the 1988 version. Therefore, no changes to shell scripts should be necessary. With this release, users should check their scripts for compatibility problems. This new version of ksh has the following functional enhancements:

- Key binding
- Associative arrays
- Complete ANSI-C printf function
- Name reference variables
- New expansion operators
- Dynamic loading of built-in commands
- Active variables
- Compound variables

For a detailed description of the new features, consult the official KornShell home page at:

`http://www.kornshell.com.`

5.4.6.3 Shell script or program calls from within C and C++ code

If your program calls shell scripts or other executables within the C or C++ source code (via the exec() system call), you should take the following considerations into account:

- Check that the shell script or program you want to call is really located at the specified directory location.
- Check that the file permissions are as you would expect them to be.
- Check that the program environment is setup correctly. For example, are all necessary environment variables set or are some environment variables set which should not be set?

- Check that the output of the called program is as you would expect it to be, especially if you are going to parse the output in your program. For example, are tabs used instead of spaces, or are data items in the expected column order?

5.5 AIX 5L porting programming tips

The following section summarizes common programming tips for writing or porting applications or device drivers to AIX 5L. They are based on the porting experience of many developers in the industry. Some are recommended porting steps, some are programming/porting practice, while some are simply guidelines for reference.

5.5.1 General tips

- Check all code with the `lint` program checker.
- Use copies of each system's commonly used header files (such as limits.h, types.h, param.h, float.h, inttypes.h, and so on) for reference. Use predefined types (including derived data types) defined in inttypes.h and types.h whenever possible.
- Use ANSI-C function prototypes so that the compiler and `lint` can help identify potential porting problems. Watch out for those function arguments that are not explicitly declared and typed. Argument sizes of such functions might not match those of the calling program.
- Decide the data cardinality of a variable before selecting the data type. Cardinality is the set of all possible values of a specific data item. Use explicit size types for cardinality-sensitive data. For example, use the uint16_t type for counting a number from 0 to 65535.
- Check the #else..#endif parts of conditional compiles. Make sure they yield the expected results.
- Define appropriate typedefs, where needed, for portability.
- Use variable argument (va_arg) lists by using va_arg() instead of declaring extra int arguments.
- Beware of compiler optimizations. Declare as volatile any variable used in a loop that is updated by an external function (for example, a signal handler). Otherwise, no changes will be detected by the loops, because the compiler will take non-volatile variables out of the loop.
- Use int for values specific to the running platform, temporary local variables, and trivial loop counters.

- Avoid storing structures that contain pointers in data files, such as database files. The address stored in the data files becomes invalid. These files then become nonportable between 32-bit and 64-bit systems, because the size of pointer has changed from 32 bits to 64 bits. One of the options is to use object references in C++ instead of pointers.
- Check for long (64 bits) bit fields. 64-bit bit fields are not supported. Structure bit fields are limited to 32 bits, and can be of type signed int, unsigned int or plain int. If you use long bit fields in 64-bit mode, their exact alignment may change in future versions of the compiler, even if the bit field is under 32 bits in length.
- Check for structure member values being passed by value in 64-bit mode. In 64-bit mode, member values in a structure passed by value to a va_arg argument may not be accessed properly if the size of the structure is not a multiple of eight bytes.
- When data alignment cannot change (for example, network packets), use `#pragma pack(1)` to avoid compiler structure padding.

5.5.2 Int, long, and pointer

- On AIX 5L, LP64 is the 64-bit programming model. Integers are 32 bits; longs and pointers are 64 bits. While under ILP32, all of them are 32 bits.
- Note that int is the default size for untyped register and unsigned variables.
- When checking or converting long declarations, check calls to printf() and appropriately change format strings to %ld, %lu, and %lx, or vice versa. Many people miss these changes during their porting, and wonder why the program produces strange output during testing.
- Check (int *) and (long *) types of casts.
- Use NULL, defined as 0L, for zero or (char *) comparisons.
- Look out for longs in unions (extra space allocated), or the storing of pointers or double values in a union whose maximum size is int.
- Network Internet addresses are 32 bits. Most network code uses longs for network addresses (16-bit leftover necessary to force 32 bits). Use unsigned int instead.
- Explicitly add a suffix (L,U, or UL) to all constants that have the potential of impacting constant assignment or expression evaluation in other parts of your program.
- Where appropriate, declare variables as int or long for alignment and performance. Do not try to save bytes by using a char or a short.

5.5.3 Sign extension

- When declaring constants, use L or UL as appropriate; when necessary, use the unsigned UL to prevent sign extension.
- Declare character pointers and character bytes as unsigned char to avoid sign extension problems with 8-bit characters.

5.5.4 Data truncation

- Avoid assigning longs to ints. Data will be truncated under the 64-bit mode.
- Avoid storing pointers in ints. Pointers will be truncated in 64-bit mode if they are assigned to ints.To avoid this problem, store pointers in variables declared as pointers or declared with intptr_t (defined in inttypes.h).
- Avoid truncating function return values. By default, the C compiler will return a value of type int if the function is undeclared. Therefore, the return value from a function may be truncated. To avoid this, use ANSI C function prototypes for user-defined functions and standard header files for C library functions and system calls.
- Use the appropriate printf()/scanf() specifiers. Make sure the algorithms that use the variable parameters are consistent with the 64-bit model. For instance, %d prints a 32-bit integer while %ld prints a 32-bit integer in 32-bit mode and a 64-bit integer (long) in 64-bit mode.
- Beware of rounding pointers for alignment (for example, & 0x03). Use (sizeof(long) - 1) for size and use appropriate casts to avoid truncating the address. For example:

  ```
  #define nround(x,y) ((((ulong)(x)+(y))-1)&~(ulong)((y)-1))
  reallen = (uchar_t *)nround(textline, LINELEN);
  ```

- In 64-bit mode, the signal() function returns a pointer (64 bits) to the previous signal handler:

  ```
  void (*signal(int signal, void (*function)( int ))) (int);
  ```

  Do not store the pointer in an int or the address will be truncated.
- When shifting bytes into a long value, ensure that each byte is cast to a long. Otherwise, the result is only a 32-bit value (not 64 bits, as expected). When shifting bits on an integer constant, specify the constant with L or UL if you want a result of type long or unsigned long, respectively. Otherwise, the results will be an integer.
- Avoid using int and long types interchangeably; it will result in truncation of significant digits or unexpected results in a 64-bit environment.

- Avoid passing long arguments to functions expecting type int. This will result in truncation.
- Avoid passing pointers to a function expecting an int type. This will result in truncation.
- Avoid assigning a long constant to an int variable. It will cause truncation without warning.
- Any function that returns a pointer should be explicitly declared when compiling in 64-bit mode. Otherwise, the compiler will assume the function returns an int and truncate the resulting pointer, even if you were to assign it to a valid pointer.
- Avoid exchanging pointers and int types; it will cause segmentation faults in the 64-bit environment when the (truncated) pointers are used.
- Avoid assignment of long types to float; it may result in loss of accuracy.
- Avoid truncating function return values. If a data type of a return value from function is incompatible with the variable to which it is assigned, the return value can be truncated. To avoid this, use ANSI C function prototypes for user-defined functions and standard header files for C library functions.

5.5.5 Data type promotion

- Avoid arithmetic between signed and unsigned numbers. Data is promoted differently in 64-bit mode than in 32-bit mode when a C program performs arithmetic operations and comparisons between signed int and unsigned long and between unsigned int and long.

5.5.6 Pointer truncation

- Use compiler option -qwarn64 or #pragma info(warn64) to help search for pointer to integer truncation.
- Avoid pointer arithmetic between long and int. In 64-bit mode, if a long value is dereferenced using an int pointer, only 32 bits will be retrieved.
- Avoid casting pointers to int or int to pointers. Such operations will give unexpected results in 64-bit mode because pointers and int variables are no longer the same size in 64-bit mode.
- Avoid storing pointers in int variables. They will be truncated in 64-bit mode.

5.5.7 Structures

- Avoid using unnamed and unqualified bit fields, as this may cause data alignment problems and/or unexpected results.
- Avoid passing invalid structure references.
- Avoid the use of undocumented or reserved bit fields.
- All the structures must be checked for alignment and size dependencies. Structure members are aligned on their natural boundaries. Structures are aligned according to the strictest aligned member. This remains unchanged from 32-bit mode. Because of the padding introduced by the member alignment, structure alignment may not be exactly the same as in the 32-bit mode. This is especially important when you have arrays of structures which contain pointer or long types.

5.5.8 Hardcoded constants

- Avoid using literals and masks that assume 32 bits.
- Avoid hardcoding the size of data types; use sizeof() instead.
- Avoid hardcoding bit shift values. Use portable defines such as LONG_BIT and WORD_BIT, which are defined in limits.h.
- Avoid hardcoding constants with malloc(). Use sizeof((void *)) to get appropriate pointer size.

5.6 AIX 5L porting guidelines

The following steps will help in the porting process.

5.6.1 Identify potential problems using grep commands

Use the `grep` or `egrep` command to help locate some potential problem areas in the source code. Use `lint` to check for programming practice in C code, which might cause problems during the 32-bit to 64-bit migration.

Text searching commands such as `grep` or `egrep` are powerful tools. They can be used to find constructs, which are likely to cause problems.

The following are some of the common potential problem areas:

- mem... functions, such as `memcpy`. The byte count might not be correct under 64-bit environment. For example:

  ```
  grep "mem*" /usr/local/src/*
  ```

- union: Look for long and int in the union. Unions that attempt to share long and int types, or overlay pointers onto int types will now be aligned differently, or be corrupted. For example:

  ```
  grep union /usr/local/src/*
  ```

- Hexadecimal constants 0[xX][0-9a-fA-F]: Look for hexadecimal constants mixed with longs or pointers. For example:

  ```
  grep 0x08001234 /usr/local/src/*
  ```

- printf()/scanf(): Look for %[dDxX]. Format codes of parameter long may need to be modified.
- sizeof(long): It no longer equals to sizeof(int).
- extern: Check the declarations of the variable.
- long: Check all long declarations. Many of them can be converted to int or unsigned int to save storage space. In the case of network addresses, if it is declared as unsigned long, replace it with unsigned int.
- Bitwise operator: (<<,>>, or ~) Add L to operands of these operators when they are long constants. This can avoid a zero result. For example:

  ```
  egrep "<<|>>|~" /usr/local/src/*
  ```

- Address operator &: Make sure the result is not stored in an int.

5.6.2 Identify potential problems using lint

Examine the problem code reported by `lint` and make necessary fixes. Please refer to the AIX 5L online command reference for the usage of `lint`.

5.6.3 Compile and link the code and fix the discovered problems

Steps such as saving compile time, link time error messages, and using debugging tools to debug the code should be commonly known practices for software developers.

A useful tip for fixing problems is: change only what is broken. This approach has several advantages:

- It minimizes the number of changes during the port, and thereby reduces the porting cycle time. This also makes the change control easier.
- It can minimize, if not avoid, the number of new problems created during the port.
- Every change is a fix of a known problem.

5.6.4 Fix alignment and padding problems

These are the core issues of porting to the LP64 model. Please refer to Section 3.10, "C and C++ data type alignment issues" on page 65 for various instances and recommendations.

5.6.5 C programming

The following issues are generally related to the C programming conventions, style, and practices.

5.6.5.1 Single source

If possible, use single source code and header files for both 32-bit and 64-bit programs.

5.6.5.2 Getting system information

Do not hard code system values in the programs. Use sysconf(), confstr(), and pathconf() to obtain the current value of certain system parameters, the configurable system limits, or whether optional features are supported, such as file implementation characteristics. Also use macros defined in limits.h to get static system limits.

5.6.5.3 Using header files

This is the best, simplest solution to make consistent changes/fixes to your source code. The following section describes the use of inttypes.h.

5.6.5.4 Using inttypes.h

Use appropriate data types in a consistent and strict manner. inttypes.h is ISO C standard header file. Use integral types defined in the header file when the applications require that you use fixed width integral types on both 32-bit and 64-bit platforms. Use int and long when the integral types need to be scalable. inttypes.h provides the following features:

- Integral type definitions
- Macros for creating constants of the types defined
- Macros for printf()/scanf() format specifiers

The following are some tips regarding usage of inttypes.h:

- Use fixed width integral types, such as int32_t for integers that must have a fixed width across platforms. For instance, use int64_t for the fixed width variable instead of long long, and uint64_t instead of unsigned long long.
- Use scalable integral types, such as long, when the date types need to scale up/down when applications moved to different platforms.

- Use the smallest integral types, if possible, to save space and control the size of the application.
- Use the intfast*n*_t data types for counters (variables that are used in frequent expression evaluation). For example, intfast32_t.
- Use a constant that matches the integer type definition. For example, use INT64_MAX with int64_t.
- intmax_t and uintmax_t should be the longest (in number of bits) signed and unsigned integer types supported by the implementation. Use them for items that must be the largest integral type.
- Use `#ifdef`s when defining fixed width 64-bit integral type variables, such as int64_t, if a single source code will be used on both platforms.
- Use the same type definition names supported by the definition of the system API you want to use.
- Data declaration of system API parameters or return values should be the same as those defined by the function prototype.

5.6.5.5 Use portable bit masks

Use scalable masks with scalable typedefs. For example:

```
#include <inttypes.h>
#ifdef __LP64__
int64_t beef=0xffffffffffffffbf
#else
int32_t beef=0xffffffbf
#endif
```

Use fixed size masks with fixed size type definitions. For example:

```
#include <inttypes.h>
int64_t refcntmask=INT64_MAX;
```

5.6.5.6 Shared memory between 32-bit and 64-bit processes

Modify the application in cases where 32-bit and 64-bit processes share the same memory segment. Check the data type of variables to be shared by these processes and modify them accordingly, if it has been determined that such mixed mode operation is inevitable, for example, in a data structure containing both data and a link list pointer. Use fixed width predefined data types in inttypes.h for data, and use int or another fixed width data type to replace the pointer, since the size of a pointer is no longer the same in 32-bit mode and 64-bit mode. The application needs to compute and then store the offset to the next structure in the fixed width variable. The application can then use both the shared memory segment base address and the offset to access the data.

5.6.5.7 Identifiers and object names

Writing software usually involves defining names for all identifiers in program modules. Identifiers may be functions, macros, variables, structures, or any object that a program module may refer to. If the software is just a simple stand-alone program, identifier naming is probably not an issue. However, if the software becomes complex and involves lots of identifiers, the following issues should be well-understood:

- Meaningful names (for the sake of debugging and maintenance): Having comments written in the source code is a good practice, but those comments will not be very useful if the source code is not available.
- Name length: With modern compilers and linkers, it is relatively safe to assume that identifiers up to 32 characters in length will be supported.

5.6.5.8 Being clear

Some programming tricks on 32-bit platforms may give different results on 64-bit platforms. For example, if you define a constant as:

```
#define ERROR 0xFFFFFFFF
```

you will find the constant to be 4,294,967,295 instead of the expected -1.

5.6.5.9 Conditional compilation

Use appropriate names for definitions used to select conditional compilation sections. Certain definitions must be used when compiling code for device drivers and kernel extensions to ensure that the correct conditional compile sections of system header files are used. If you are writing a device driver or kernel extension, it makes sense to use the same variables for conditional code selection in your driver code. If you are writing a user-level application, it may be possible to use one of the symbols automatically defined by the compiler as the value for selecting conditional sections.

5.6.5.10 Use macros

Macros also help improve the readability of your code if their names are meaningful. For example:

```
#define MAXITEMNUM 2500
...
    if (i > MAXITEMNUM) report_it(i);
...
```

is more meaningful than:

```
1      if (n > 2500) report_it(i);
```

You can also use macros to avoid hardcoding constants, especially commonly-used values. The following are some examples of using macros:

- Use itemno & ((1<<(8*sizeof(int)))-1) instead of itemno & 0xFFFF.
- Use malloc(MAXITEMNUM*sizeof(int)) instead of malloc(400).
- Use char buf[PAGE_SIZE] instead of char buf[4096].

Note that macros are processed by the C preprocessor, which is less informative than the compiler. If you need a warning about type misuse of a constant, define the constant as a variable with the keyword const rather than defining it as a macro with `#define`.

5.6.5.11 In-line assembler language code

There is no support for coding in-line assembler within C source code on AIX 5L. Assembly code must be written as callable functions contained in assembler source files. These are then assembled to create object files.

5.6.5.12 Use stdarg.h for variable argument functions

The layout in memory of parameters (or arguments) to function calls is completely up to the compiler and will be transparent as long as those parameters are referred to by name. There may be cases, however, where there is a need to indirectly refer to a parameter through a reference to another in the list. One example is printf style functions. Another example is the *variable argument function*, an example of which is shown in Figure 27.

```
1   long lsum(int num, long val, ...)
2   {
3       long l, *p = &val;
4       for (l = val; num > 1; num--) { l += *++p; }
5       return (l);
6   }
7
8   int main(void) {
9       printf("total %ld\n", lsum(4, 8L, 4L, 7L, 2L));
10      exit(0);
11  }
```

Figure 27. Example of variable parameter function

In line 4, a particular parameter layout in the function call stack is assumed. This assumption may cause problems when porting to a platform with a different stack layout. The portable way to do this is to use the functionality provided by the ANSI C defined header file, <stdarg.h>, as shown in Figure 28. This is particularly important on the Itanium platform, where variables of type char, short, and int are less than 64 bits in size, they are zero extended to 64 bits before being passed as arguments to procedures. This is because the stack is implemented using 64-bit registers rather than system memory.

```
#include <stdarg.h>
long lsum(int num, long val, ...)
{
    long l;
    va_list p;
    va_start(p, val);
    for (l = val; num > 1; num--) { l+= va_arg(p, long); }
    return (l);
}

int main(void) {
    printf("total %ld\n", lsum(4, 8L, 4L, 7L, 2L));
    exit(0);
}
```

Figure 28. Example of <stdarg.h> usage

Chapter 6. Makefiles and the make command

In this chapter, we will examine the `make` command and the supported features of makefiles on AIX 5L, and compare them with the supported features of HP-UX, Solaris, Tru64 and GNU `make` command. We will concentrate on the features that are found across all the platforms. The chapter is targeted at people who use several more advanced features in their makefiles and in their use of the `make` command. If you are only making simple makefiles such as the one shown in Figure 29, and do not plan to do otherwise, you might want to continue onto the next chapters.

Not all details of makefiles and the `make` command are covered in this chapter; that is beyond the scope of this redbook. However, this chapter covers the most commonly used features.

```
foobar: foo.o bar.o
      cc -o foobar foo.o bar.o

foo.o: ./src/foo.c ./inc/foo.h
      cc -c ./src/foo.c

bar.o: ./src/bar.c ./inc/bar.h
      cc -c ./src/bar.c

clean:
      rm -f *.o
      rm -f foobar
```

Figure 29. A very simple makefile

If you are only using the `make` command with the target as an argument, as shown in Figure 30 on page 156, you also might want to skip this chapter and continue with the next one.

```
$ make clean
        rm -f *.o
        rm -f *.a
        rm -f foobar
$ make foobar
        cc -c ./src/foo.c
        cc -c ./src/bar.c
        cc -o foobar foo.o bar.o
$
```

Figure 30. Simple use of the `make` *command*

If you are currently using GNU `make` and would prefer to continue to use GNU `make` on AIX 5L, you might also want to read quickly through this chapter. Pay particular attention to Section 6.2, “The make command” on page 169, to ensure that you have installed GNU `make` and that you are actually using it, rather than the default AIX 5L `make` command.

6.1 Makefiles

Here we will first discuss what a makefile contains: rules, macro definitions and comments. There are two types of rules: inference rules and target rules. The `make` command reads from a file that contains a set of build-in inference rules.

Comments start with the pound sign (#) and continue until an un-escaped <newline> is reached.

6.1.1 Command prefixes

Command lines can have one or more prefixes; these function in the same way on all the covered platforms. Below is a description of how these prefixes work.

@ If a command has the @ (at) character in front of it, the command will not be written to standard output. If the -n command line option has been specified with `make`, the command will be echoed, thus -n overrides the @. If the -s option has been specified on `make`, or the .SILENT rule has been specified with no prerequisites, all commands will not be echoed. Another way of achieving the same effect is to put the current target as a prerequisite to the .SILENT rule.

- If a command has the - (hyphen) character in front of it, any error from the command being executed will be ignored. If the -i option has been given to the `make` command, or the .IGNORE rule has been specified with no prerequisites, all errors will be ignored. Another way of achieving the same effect is to put the current target as a prerequisite to the .IGNORE rule.
+ If a command has the + (plus sign) in front of it, the command will be executed, even though the options -n, -q or -t are specified.

6.1.2 Default inference rules

Default inference rules govern what the `make` command does by default when encountering a rule like the foo.o rule shown in Figure 31. This is a very simple makefile, where the building of foo.o and bar.o relies on the default inference rule for `o.c`, or, in other words, the default way to build a <name>.o file from a <name>.c file.

```
$ ls
bar.c     foo.c     makefile
$ cat makefile
foo.o: foo.c ../inc/foo.h

bar.o: bar.c ../inc/bar.h

foobar:  foo.o bar.o
        cc -o foobar foo.o bar.o
$
```

Figure 31. A simple makefile that uses the default .o.c inference rule

Now, when we try to build the foobar program, as in Figure 32, we see the rule that the `make` command uses to build the foo.o and bar.o files:

```
$(CC) $(CFLAGS) -c $<
```

Where the $(CFLAGS) equals -O.

```
 make foobar -f makefile.2
        cc -O -c foo.c
        cc -O -c bar.c
        cc -o foobar foo.o bar.o
$
```

Figure 32. Make which uses the default inference rule .o.c to build foobar

There are some potential problems in using the default inference rules, particularly when moving makefiles to another platform. The reason for this is that the default inference rules are not the same on all platforms. As an example, Table 51 compares the default rules for .c.a (creating an ar format archive file from a .c source file) on AIX and Tru64.

Table 51. The .c.a inference rules

Platform	Rule

AIX	`$(CC) -c $(CFLAGS) $<`
	`$(AR) $(ARFLAGS) $@ $*.o`
	`rm -f $*.o`

Tru64	`$(CC) $(CFLAGS) -c $<`
	`$(AR) $(ARFLAGS) $@ $*.o`
	`$(RM) $(RMFLAGS) $*.o`

Although the differences are minor in this example, this is not always the case. For example, if we have a makefile like the one shown in Figure 33, we might run into trouble.

```
RM=./domove

foo.o: foo.c ../inc/foo.h
   $(CC) -c $*.c

bar.o: bar.c ../inc/bar.h
   $(CC) -c $*.c

bar.a: bar.o

foo.a: foo.o

foobar: ./obj/foo.o ./obj/bar.o
   $(CC) -o ./obj/foobar $?
   strip ./obj/foobar
clean:
   rm -f ./obj/*.o
   rm -f *.a
   rm -f ./obj/foobar
```

Figure 33. Makefile that uses the default rule on how to make .o from .a files

As you see RM has been set to ./domove which is shown in Figure 34 on page 159, along with an `ls` listing of the files used. This script simply moves the .o files to ./obj/, where they are used to build the foobar program. This makefile will work on Tru64; look at Table 51 to figure out why.

```
$ ls
bar.c     domove    foo.c     makefile  obj
$ cat domove
mv $1 ./obj/$1
$ ls obj
$
```

Figure 34. Source files and the domove script

On AIX 5L, on the other hand, you will run into problems, due to the fact that the default rule for making a .a file from a .c file is different. The output of what happens in that case can be seen in Figure 35.

```
$ make clean
        rm -f ./obj/*.o
        rm -f *.a
        rm -f ./obj/foobar
$ make foo.a
        cc -c foo.c
        cc -c -O foo.c
        ar -rv foo.a foo.o
ar: Creating an archive file foo.a.
a - foo.o
        rm -f foo.o
$ make bar.a
        cc -c bar.c
        cc -c -O bar.c
        ar -rv bar.a bar.o
ar: Creating an archive file bar.a.
a - bar.o
        rm -f bar.o
$ make foobar
make: 1254-002 Cannot find a rule to create target ./obj/foo.o from dependencies.
Stop.
$
```

Figure 35. Running the makefile from Tru64 on AIX 5L

As you can see, the `make` command could not find the ./obj/foo.o file. This is due to that fact that the AIX 5L `make` command does not use $(RM) in the default rule on how to make a .a file from a .c file. It uses the `rm` command instead (this is the POSIX way of making a .a file from a .c file).

6.1.3 Single suffix default inference rules

There are two kinds of default rules: single suffix and double suffix. The single suffix rules govern how to generate an executable. An executable has no file name suffix, and is generated from a <filename>.<suffix> file where <suffix>, for example, can be a c for a C source file or a C for a C++ source file.

To have a complete overview of the single suffix default inference rules that are defined on various UNIX platforms, refer to Table 110 on page 477.

You should search your makefiles, looking for instances where you assume that single suffix rules work in a particular way.

6.1.4 Double suffix default inference rules

The double suffix default rules are the rules that cover how to generate a <filename>.<suffixa> from a <filename>.<suffixb> file. The rules are listed in Table 111 on page 491.

As with the single suffix rules, you should search your makefiles looking for places where you assume that double suffix rules work in a particular way.

6.1.5 Special targets (the .targets)

Special targets are called by different names on the various platforms:

AIX 5L	Special targets
HP-UX	Built-In targets
Solaris	Special-Function targets
Tru64	Pseudotarget names and Special targets
GNU	Special built-in targets
POSIX	Special targets

Even if the name for the feature is not the same, the functions are. Special targets are targets that have a special meaning. If you do not use any special targets in your makefiles, you can skip this section. If you want to make sure that your makefiles do not use special targets, try running the shell script that is shown in Appendix A.1.1, “The find_spec_targets_aix.ksh sample program” on page 451. This script will find the special targets in your makefiles that are not supported by the AIX 5L `make` command. There is also a script in Appendix A.1.2, “The find_spec_targets_gnu.ksh sample program” on page 452 that will find the special targets that are not supported by the GNU `make` command.

If you already have a pretty good idea which special targets you are using, you should have a look in Table 52 and check to see if you are currently using any special targets that are not supported by the AIX 5L `make` command.

Table 52. Comparison of special target support

Special target	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
.DEFAULT	yes	yes	yes	yes	yes	yes
.DELETE_ON_ ERROR	no	no	no	no	yes	no
.DONE	no	no	yes	no	no	no
.EXIT	no	no	no	yes	no	no
.EXPORT_ALL_ VARIABLES	no	no	no	no	yes	no
.FAILED	no	no	yes	no	no	no
.GET_POSIX	no	no	yes	no	no	no
.IGNORE	yes	yes	yes	yes	yes	yes
.INIT	no	no	yes	yes	no	no
.INOBJECTDIR	no	no	no	yes[4]	no	no
.INTERMEDIATE	no	no	no	no	yes	no
.INTERRUPT	no	no	no	yes[3]	no	no
.KEEP_STATE	no	no	yes	no	no	no
.KEEP_STATE_FILE	no	no	yes	no	no	no
.MAIN	no	no	no	yes[3]	no	no
.MAKE_VERSION	no	no	yes	no	no	no
.MUTEX	no	yes	no	no	no	no
.NOTPARALLEL	no	no	no	no	yes[1]	no
.NO_PARALLEL	no	no	yes[1]	no	no	no
.PARALLEL	no	no	yes[2]	no	no	no
.PATH	no	no	no	yes[3]	no	no
.PATHsuffix	no	no	no	yes[3]	no	no
.PHONY	no	no	no	no	yes	no

Special target	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
.POSIX	yes	no	yes	yes[3]	no	yes
.PRECIOUS	yes	yes	yes	yes	yes	yes
.SCCS_GET	no	no	yes	yes[3]	no	no
.SCCS_GET_POSIX	no	no	yes	no	no	no
.SECONDARY	no	no	no	no	yes	no
.SILENT	yes	yes	yes	yes	yes	yes
.SUFFIXES	yes	yes	yes	yes	yes	yes
.WAIT	no	no	yes	no	no	no
yes[1] These two special targets have the same effect. yes[2] Reserved for future use, has no effect. yes[3] Only available in the POSIX version of make. yes[4] Only available in the standard make.						

If you should find that you are using some special targets that are not supported on AIX 5L, then the best approach is to get your makefile to work on your source platform without the use of the special target that is not supported by the AIX 5L `make` command. In most cases, this should not pose a problem.

6.1.6 Using the .POSIX special target

The .POSIX special target can be used to make your makefile POSIX compliant. This is not as simple as it sounds. You will still have to make sure that your makefile does what it is supposed to do. But it does give you the ability to make most of your porting on your source platform.

If you only want to use POSIX compliant features in your makefiles, the .POSIX special target must be entered in the top of your makefile, as the first non comment line to take effect.

If you are using HP-UX or GNU `make`, even if there is a .POSIX in your makefile (because neither `make` command supports the .POSIX special target), it will *not* make the makefile behave like a POSIX compliant makefile. The makefile will still be executed, but you will *not* receive an error message.

To test if your `make` command supports the .POSIX special target, you should examine a predefined macro that has one value when you run with .POSIX and another when you are running without it. As an example, the $(CC) macro is a good candidate, because it is defined as c89 when you use the .POSIX special target, and, on most platforms, the $(CC) macro has the value cc when you are not using .POSIX. You can also have a look in Table 52 on page 161 for more inspiration.

An example on how to test to see if the `make` command supports the .POSIX special target can be seen in Figure 36.

```
$ cat > makefile
.POSIX:

posixtest:
        @echo $(CC)

$ make posixtest
c89
$ cat makefile.non
posixtest:
        @echo $(CC)

$ make -f makefile.non posixtest
cc
$
```

Figure 36. Test to see if make supports the .POSIX special target

So if you normally have a .POSIX as the first non-comment line in your makefile, you should be able to use your makefiles on AIX 5L without any major rewriting. You will still have to check that any shell commands from within the makefile are executed correctly.

6.1.7 Internal macros

Internal macros are called by different names on the various platforms. The following names are used for Internal macros:

AIX 5L	Internal macros
HP-UX	Built-in macros
Solaris	Dynamic macros
Tru64	Internal make macros
GNU	Automatic variables
POSIX	Internal macros

Throughout this chapter, we will use the term used by AIX 5L, internal macros. Even tough internal macros go by different names on the different platforms; the way they function in your makefiles are more or less the same. Their function is listed in the following paragraphs, so that you can compare if the way the internal macro works on AIX 5L is the same as on your platform.

There are POSIX defined internal macros; GNU `make` has defined two more, which are called $+ and $^.

$@ Will be evaluated to the full target name of the current target or the archive file name part of a library archive target. This rule is evaluated both for target and inference rules.

For example, in:

```
foobar: foo.o bar.o
    cc -o $@ foo.o bar.o
```

$@ will be evaluated to foobar

$% Will be evaluated only when the current target is an archive library member of the form libraryname(libmember.o). So when the target is an archive library, $@ will be evaluated to libraryname and $% will be evaluated to libmember.o. This rule is evaluated both for target and inference rules. In Figure 37 on page 165, you can see the how to use $@ and $%.

```
F_EX = foo_add.exp
B_EX = bar_add.exp
EXPORTFILES = $(F_EX) $(B_EX)

foo.o: ./src/foo.c ./inc/foo.h
      cc -c ./src/foo.c

bar.o: ./src/bar.c ./inc/bar.h
      cc -c ./src/bar.c

lib_foobar.a(lib_foo.o): foo.o $(EXPORTFILES)
      cc -o $% foo.o -bE:$(F_EX) -bI:$(B_EX) -bM:SRE -bnoentry

lib_foobar.a(lib_bar.o): bar.o $(EXPORTFILES)
      cc -o $% bar.o -bE:$(B_EX) -bI:$(F_EX) -bM:SRE -bnoentry

lib_foobar.a: lib_foobar.a(lib_foo.o) lib_foobar.a(lib_bar.o)
      ar rv $@ lib_foo.o
      ar rv $@ lib_bar.o
```

Figure 37. A makefile that uses $% and $@

The first occurrence of $% in Figure 37 will be evaluated to lib_foo.o and the second occurrence of $% will be evaluated to lib_bar.o. The two occurrences of $@ will both be evaluated to lib_foobar.a.

$< Will be evaluated to the file name whose existence allowed the inference rule to be chosen for the target. When used in the .DEFAULT rule, the macro will be evaluated to the current target name. This rule is evaluated *only* for inference rules.

For example, in:

```
foo.o: foo.c
    cc -c $<
```

$< will be evaluated to foo.c

$? Will be evaluated to a list of prerequisites that are newer than the current target. This rule is evaluated both for target and inference rules.

For example, in:

```
foobar: foo.o bar.o
    cc -o foobar $?
```

$? will be evaluated to foo.o bar.o

$^	Will be evaluated to the names of all the prerequisites, with spaces between them. For prerequisites that are archive members, only the member name is used. Duplicate prerequisites are omitted.

For example, in:

```
OBJFILES = foo.o bar.o bar.o foo.o
foobar:  $(OBJFILES)
    cc -o foobar $+
```

$+ will be evaluated to foo.o bar.o.

$+	Will be evaluated to the names of all the prerequisites, with spaces between them. For prerequisites that are archive members, only the member name is used. Duplicate prerequisites are all retained and in a preserved order.

For example, in:

```
OBJFILES = foo.o bar.o bar.o foo.o
foobar:  $(OBJFILES)
    cc -o foobar $^
```

$+ will be evaluated to foo.o bar.o bar.o foo.o

$$@	Will be evaluated to the label name from the left side of the dependency line. This rule is evaluated *only* for target rules.

For example, in:

```
TARGET_CMDS = foo bar foobar
$(TARGET_CMDS) : $$@.c
    cc $? -o $@
```

$$@ will first be evaluated to foo, then bar, and then foobar

$(F)	When F is appended to one of the above macros in the form $(%F) or $(*F), the macro is evaluated the file name part of the macro.
$(D)	When D is appended to one of the above macros, in the form $(%D) or $(*D), the macro is evaluated the directory part of the macro.

Table 53 lists the internal macros supported by different UNIX platforms.

Table 53. Internal macro support

Special macro	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
$*	yes	yes	yes	yes	yes	yes
$%	yes	yes	yes	yes[a]	yes	yes
$?	yes	yes	yes	yes	yes	yes
$<	yes	yes	yes	yes	yes	yes
$@	yes	yes	yes	yes	yes	yes
$^	no	no	no	no	yes	no
$+	no	no	no	no	yes	no
$$@	yes	yes	yes	yes	yes	yes
$(F)	yes	yes	yes		yes	yes
$(D)	yes	yes	yes		yes	yes

a. Only available in the POSIX version of `make`.

By looking at Table 53, it is clear that the same internal macros are implemented on the different `make` commands, except for the $+ and $^, which are unique to the GNU `make` command. There should not be any problems in porting this feature of the makefile to AIX 5L. You simply do not need to change anything.

Appendix A.1.5, "The find_internal_macro_aix.ksh sample program" on page 455 lists a small shell script that will search your source tree and try to find makefiles that contain the $^ and $+ strings.

6.1.8 Predefined macros

Predefined macros are variables that have a predefined value. A list of which predefined macros there are on the various platforms can be seen in Table 54 on page 168. The reason why a makefile might stop to function *correctly* when moved to another machine is that the macros are not necessarily predefined (or may be defined differently) on the new platform.

Table 54. Predefined macros

Predefined macro	**AIX 5L**	**HP-UX**	**Solaris**	**Tru64**	**GNU**	**POSIX**
AR	ar	-	ar	ar	ar	ar
AS	as	as	as	as	as	-
CC	cc	cc	cc	cc	cc	c89
CCC	xlC	-	$CC	-	-	-
CO	co	-	-	co	co	-
CP	-	-	-	cp	-	-
CPP	$(CC) -E	-	-	-	$(CC) -E	-
CXX	-	CC	-	-	g++	-
CWEAVE	-	-	-	-	cweave	-
CTANGLE	-	-	-	-	tangle	-
EC	-	-	-	efl	-	-
F77	-	-	-	-	$(FC)	-
FC	xlf	f77	f77/f90	f77	xlf[a]	fort77
GET	get	get	-	-	get	-
LD	ld	ld	ld	ld	ld	-
LEX	lex	lex	lex	lex	lex	lex
LINT	-	-	lint	lint	lint	-
MACHINE	-	-	-	alpha	-	-
MAKE	make	make	-	make	$(MAKE _COMM AND)	make
MAKE_ COMMAND	-	-	-	-	make	-
MAKEFILE	-	-	-	makefile	-	-
MAKEINFO	-	-	-	-	makeinfo	-
MV	-	-	-	mv	-	-

Predefined macro	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
M2C	-	-	m2c	-	m2c	-
PC	-	pc	pc	pc	pc	-
RANLIB	-	-	-	ranlib	-	-
RC	-	-	-	f77	-	-
RM	-	-	rm -f	rm	rm -f	-
RMFLAGS	-	-	-	-f	-	-
TANGLE	-	-	-	-	tangle	-
TEX	-	-	-	-	tex	-
TEXI2DVI	-	-	-	-	texi2dvi	-
WEAVE	-	-	-	-	weave	-
YACC	yacc	yacc	yacc	yacc	yacc	yacc
YACCE	-	-	-	yacc -e	-	-
YACCR	-	-	-	yacc -r	`yacc -r`	-

a. The manual claims that FC is set to f77, but if you examine the output from `make -p` on AIX 5L, it is set to xlf.

If you want to check your makefiles, there is a script in Appendix A.1.3, "The find_predef_macro_aix.ksh sample program" on page 453 that will find those makefiles that contain predefined macros that are not supported by the AIX 5L `make` command. If you are planning on using GNU `make` to build your programs, there is a version of the script in Appendix A.1.4, "The find_predef_macro_gnu.ksh sample program" on page 454. These scripts will find makefiles that includes predefined macros that are not supported by the `make` command in question.

If you need to use a predefined macro that is not defined on AIX 5L, you can either define it when using the command line or simply define it inside your makefile.

6.2 The make command

The `make` command operates on makefiles, or if no makefiles are present, the `make` command will use built-in values to build the specified target. An example of this can be seen in Figure 38 on page 171.

The `make` command will try to find a makefile by trying out different names and different locations; the places and names of makefiles is a little different from platform to platform. In Table 55, you can see the search list of the different `make` commands. The number specified in the table is the order in the search list of that particular makefile. Thus, 1 means that this is the makefile searched for first, 2 means that this is the makefile searched for second, and so on. You can use the -f option to the `make` command to control which makefiles are used.

Table 55. Search list for makefiles for the different make commands

Makefile	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
./makefile	1	1	1	1	1	1
./Makefile	2	2	2	2	2	2
./SCCS/.makefile	3	5	4	-	6	-
./SCCS/.Makefile	4	6	6	-	8	-
./s.makefile	-	3	3	-	5	-
./s.Makefile	-	4	5	-	7	-
./makefile,v	-	-	-	3	3	-
./Makefile,v	-	-	-	4	9	-
./RCS/makefile.v	-	-	-	5	4	-
./RCS/Makefile.v	-	-	-	6	10	-

Be sure that you check your source tree for makefiles, especially if you are using the SCCS (source code control system) or the RCS (revision control system). Have a look at Table 55 to make sure that you are using the right makefiles in the right order.

Because of the default suffix rules, you can actually build targets without a makefile. An example of this can be seen in Figure 38 on page 171.

```
$ ls
bar.c  foo.c
$ make bar.o foo.o
        cc -O -c bar.c
        cc -O -c foo.c
$ ls
bar.c  bar.o  foo.c  foo.o
$
```

Figure 38. make used without a makefile

So, for example, if you expect to *fetch* a makefile from RCS, which the `make` command under AIX 5L does not support, your code might still compile due to the fact that there are default suffix rules. This might have unexpected results.

Depending on which software you have installed on your machine, and on which platform, you can have more than one `make` command installed. On Tru64, you might have three different `make` commands installed:

- /usr/bin/posix/make
- /usr/opt/ultrix/usr/bin/make
- /usr/bin/make

Furthermore, you might even have installed GNU `make`, which normally will be situated in /usr/local/bin/make. But the location of the GNU `make` might be in a totally different directory; it all depends on how it was installed and by who.

You can use the `which` command to determine the `make` command you are using, as shown in Figure 39 on page 172. In this example, the `make` command that appears first in the path is the standard AIX 5L `make` command.

```
$ which make
/usr/bin/make
$ echo $PATH
/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/local/
bin:.
$ export PATH=/usr/local/bin/:$PATH
$ echo $PATH
/usr/local/bin/:/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/
sbin:/usr/local/bin:.
$ which make
/usr/local/bin/make
$
```

Figure 39. Verifying which make you are using and changing to GNU make

6.2.1 Environment variables

Both the LANGUAGE and PATH environment variables impact how your makefile operates. If you have more than one `make` command installed on your system, it is nice to know which `make` command you are actually executing. To check this, see Section 6.1.1, “Command prefixes” on page 156.

For information on how the language environment affects your program and the `make` command, look in Chapter 10, “National Language Support” in *System Management Guide: Operating System and Devices*, which can be found in the AIX online documentation.

The other environment variables that affect or are affected by the `make` command are listed in Table 56.

Table 56. Environment variables and the make command

Variable	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
KEEP_STATE	no	no	yes	no	no	no
MAKECWD	no	no	no	yes	no	no
MAKEFLAGS	yes	yes	yes	yes	yes	yes
MAKEPSD	no	no	no	yes	no	no
MFLAGS	yes	no	no	no	no	no
OBJECTDIR	no	no	no	yes	no	no
SOURCEDIR	no	no	no	yes	no	no

Variable	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
USE_SVR4_ MAKE	no	no	yes	no	no	no
VPATH	yes	yes	no	yes	yes	no

6.2.2 Command line options to the make command

The `make` command understands several command line options, which are not the same from platform to platform. Check in Table 57 that you are not using any options on your source platform that are not supported on AIX 5L.

Table 57. Switches used by the different make commands

Option	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
-b	no	yes	no	yes[a]	yes	no
-c	no	no	no	yes	no	no
-d	yes	yes	yes	no	yes	no
-dd	no	no	yes	no	no	no
-e	yes	yes	yes	yes	yes	yes
-f	yes	yes	yes	yes	yes	yes
-h	no	no	no	no	yes	no
-i	yes	yes	yes	yes	yes	yes
-j	no	no	no	no	yes	no
-k	yes	yes	yes	yes	yes	yes
-l	no	no	no	no	yes	no
-m	no	no	no	yes[b]	yes	no
-n	yes	yes	yes	yes	yes	yes
-o	no	no	no	no	yes	no
-p	yes	yes	yes	yes	yes	yes
-q	yes	yes	yes	yes	yes	yes
-r	yes	yes	yes	yes	yes	yes
-s	yes	yes	yes	yes	yes	yes
-t	yes	yes	yes	yes	yes	yes

Option	AIX 5L	HP-UX	Solaris	Tru64	GNU	POSIX
-u	no	yes	no	yes	no	no
-v	no	no	no	no	yes	no
-w	no	yes	no	no	yes	no
-x	no	no	no	yes	no	no
-y	no	no	no	yes[c]	no	no
-B	no	yes	no	no	no	no
-C	no	no	no	yes	yes	no
-D	yes	no	yes	no	no	no
-DD	no	no	yes	no	no	no
-F	no	no	no	yes[d]	no	no
-I	no	no	no	no	yes	no
-K	no	no	yes	no	no	no
-N	no	no	no	yes	no	no
-P	no	yes	yes	no	no	no
-R	no	no	no	no	yes	no
-S	yes	yes	yes	yes[e]	yes	yes
-Tp	no	no	no	yes[f]	no	no
-U	no	no	no	yes	no	no
-V	no	no	yes	no	no	no
-W	no	no	no	no	yes	no

a. Only available in the standard version of `make`.
b. See Footnote a.
c. See Footnote a.
d. See Footnote a.
e. See Footnote a.
f. See Footnote a.

6.2.2.1 make options on HP-UX

Here we will mention the options that are different on the HP-UX `make` command, compared to the `make` command on AIX 5L, and also give a short explanation on what to do to port these options.

-b Turns on compatibility mode for old Version 7 makefiles. If you encounter a -b option in any scripts, just remove it.

-u Will make all targets, and ignore all timestamps. If you need this functionality, make a `clean` target that removes the files that need to be removed to enable a full remake, then use a `make clean` before you attempt to rebuild all targets.

-w Will suppress warning messages, not fatal messages. If you are using `make` from a script where a warning message is not desired, then use the `grep` command to suppress the warning messages.

-B Turns off compatibility mode for old Version 7 makefiles. If you encounter a -B option in any scripts, just remove it.

-P Will turn on parallel compilation of the program. The AIX 5L `make` command does not support a make that runs in parallel. With the very fast machines that are available today compilation time is not normally a problem. If it is, then you can build a script that can control a parallel execution of several `make` commands. An example of how this could be implemented is shown in Figure 40. Alternatively, consider using GNU `make`.

```
#!/usr/bin/ksh
cd /home/jesper/test/src/foo/
make foo.o&
WAITING="$WAITING $!"
cd /home/jesper/test/src/bar/
make bar.o&
WAITING="$WAITING $!"
wait $WAITING
cd /home/jesper/test/src/
make foobar
```

Figure 40. Using a shell script to obtain a parallel make

6.2.2.2 make options on Solaris

Here we will mention the options that are different on the Solaris `make` command compared to the `make` command on AIX 5L. We also give a short explanation on what to do to port these options.

-dd This option displays the dependency check and processing to port this option use the -d flag. If it does not show enough information, you will need to look in your makefiles.

-DD Displays the text of the makefiles, make rules, the state file, and all hidden dependency reports. To port this option, you can use the -p option, which might give you what you are looking for, but the best way is still to look in your makefiles and in your $MAKERULES file.

-K Uses the MAKERULES macro that is described in Section 6.2.3, “The MAKERULES macro on make for AIX 5L” on page 178.

-P This option displays dependencies rather than making them. There really is no equivalent for this option. What you will have to do is look in your makefiles to see how the dependencies are defined.

-V Puts `make` into system V mode. If you are using this option on Solaris, you should first port your makefiles to run without it before proceeding.

6.2.2.3 Make options on Tru64

Here we will mention the options that are different on the Tru64 `make` command compared to the `make` command on AIX 5L, and also give a short explanation on what to do to port these options.

-b This option has no effect. On Tru64, it exists so that older versions of make dependency files continue to work. If you are using this option on Tru64, you should first port your makefiles to run without needing it before proceeding.

-c Does not try to make a corresponding RCS file and check it out if the file does not exist.

-m Searches the machine-specific subdirectories first. To port this option, you simply have to create your own MACHINE macro, and then use a $(MACHINE) in the your path to your source files. Look at Figure 41 and Figure 42 on page 177 for examples on how to do a port.

```
OBJFILES = foo.o bar.o

foobar:  $(OBJFILES)
    $(CC) -qarch=$(MACHINE)-o foobar $?

foo.o: ./$(MACHINE)/foo.c ./inc/foo.h
    $(CC) -c -qarch=$(MACHINE) $(<D)/$*.c

bar.o: ./$(MACHINE)/bar.c ./inc/bar.h
    $(CC) -c -qarch=$(MACHINE) $(<D)/$*.c

clean:
    rm -f *.o
    rm -f *.a
    rm -f foobar
```

Figure 41. Simple makefile that uses a variable in the PATH to the source files

```
$ ls
#makefile#  inc         makefile~   pwr2        pwr4        tmp
SCCS        makefile    pwr         pwr3        src
$ make MACHINE=com foobar
        cc -c -qarch=com ./com/foo.c
        cc -c -qarch=com./com/bar.c
        cc -qarch=com -o foobar foo.o bar.o
$ make clean
        rm -f *.o
        rm -f *.a
        rm -f foobar
$ make MACHINE=pwr3 foobar
        cc -c -qarch=com ./pwr3/foo.c
        cc -c -qarch=com ./pwr3/bar.c
        cc -qarch=com -o foobar foo.o bar.o
$ make clean
        rm -f *.o
        rm -f *.a
        rm -f foobar
$ make MACHINE=pwr4 foobar
        cc -c -qarch=com ./pwr4/foo.c
        cc -c -qarch=com ./pwr4/bar.c
        cc -qarch=com -o foobar foo.o bar.o
$
```

Figure 42. Using $(MACHINE) to build programs for different implementations

-u Does not unlink files that were previously checked out by RCS.

-x Does not execute any commands.

-y Checks target files for dependencies.

-C Tries to find a corresponding RCS file and checks it out if the file does not exist.

-F Causes a fatal error to occur if the description file is not present. To port this functionality, you can always put in a piece of code in your makefile that does the check for you.

-N Disables all Makeconf processing.

-U Unlinks files that were previously checked out by RCS. This is the opposite of the -u option.

6.2.2.4 Make options on GNU make

If you are using GNU `make` on your source platform, the easiest way to port your `make` options is to keep using GNU `make`, due to the fact that it runs perfectly well on AIX 5L.

6.2.3 The MAKERULES macro on make for AIX 5L

There is yet another way to port your makefiles to AIX 5L. The `make` command on AIX 5L supports a macro called MAKERULES. The value of this macro contains the name of the rule file for the `make` command to use. The default value of the MAKERULES macro is:

```
/usr/ccs/lib/aix.mk
```

When the .POSIX macro is defined, the value of the MAKERULES macro is:

```
/usr/ccs/lib/posix.mk
```

If you look in Table 57 on page 173, you can see the switches supported by the `make` command on the different platforms. One switch that is supported on all the `make` commands is the -p flag. This flag instructs the `make` command write to standard output the complete set of macro definitions and target descriptions. Thus if you do a `make -p > <sourceos>.mk`, this will create a file that contains the complete set of macro definitions and target descriptions for the particular `make` command on the operating system you are using.

You can transfer this file to your AIX 5L development environment. Now you can try to run the AIX 5L `make` command with the MAKERULES macro set to the file you just transferred, and thus have a make/makefile environment similar to your source platform. You will have to do modifications to your new .mk file to get it to work.

There is an example of a makefile being evaluated using the makerules from Solaris 8 in Figure 44 on page 180. The makefile used can be seen in Figure 43 on page 179.

```
F_EX = foo_add.exp
B_EX = bar_add.exp
EXPORTFILES = $(B_EX) $(F_EX)
OBJFILES = foo.o bar.o

foobar:  $(OBJFILES)
        $(CC) -o foobar $?

foo.o: ./src/foo.c ./inc/foo.h
        $(CC) -c $(<D)/$*.c

bar.o: ./src/bar.c ./inc/bar.h
        $(CC) -c $(<D)/$*.c

lib_foobar.a(lib_foo.o): foo.o $(EXPORTFILES)
        $(CC) -o $% foo.o -bE:$(F_EX) -bI:$(B_EX) -bM:SRE -bnoentry

lib_foobar.a(lib_bar.o): bar.o $(EXPORTFILES)
        $(CC) -o $% bar.o -bE:$(B_EX) -bI:$(F_EX) -bM:SRE -bnoentry

lib_foobar.a:lib_foobar.a(lib_foo.o) lib_foobar.a(lib_bar.o)
        $(AR) rv $@ lib_foo.o
        $(AR) rv $@ lib_bar.o
clean:
        rm -f *.o
        rm -f *.a
        rm -f foobar
```

Figure 43. Makefile used with solaris.mk and hpux.mk files

This method is by no means a solution that will instantly port your makefiles to AIX 5L, but it does enable you to retain some functionality from your source development platform on your AIX 5L development platform. You can use this functionality to either do some of your porting on your new platform or simply use the old .mk files as a part of your development environment.

```
$ make clean
        rm -f *.o
        rm -f *.a
        rm -f foobar
$ make lib_foobar.a
        cc -c ./src/foo.c
        cc -o lib_foo.o foo.o -bE:foo_add.exp -bI:bar_add.exp -bM:SRE -bnoentry
        cc -c ./src/bar.c
        cc -o lib_bar.o bar.o -bE:bar_add.exp -bI:foo_add.exp -bM:SRE -bnoentry
        ar rv lib_foobar.a lib_foo.o
ar: Creating an archive file lib_foobar.a.
a - lib_foo.o
        ar rv lib_foobar.a lib_bar.o
a - lib_bar.o
$ ls -l *.a
-rw-r--r--   1 jasper   usr           3757 Mar 05 13:28 lib_foobar.a
$ make MAKERULES=/usr/ccs/lib/solaris.mk clean
"./solaris.mk", line 402: make: 1254-052 Variable not specified correctly: $

make: 1254-058 Fatal errors encountered -- cannot continue.
$ ed solaris.mk
8503
402
$= $
d
w
8498
q
$ make MAKERULES=/usr/ccs/lib/solaris.mk clean
        rm -f *.o
        rm -f *.a
        rm -f foobar
$ make MAKERULES=/usr/ccs/lib/solaris.mk lib_foobar.a
        cc -c ./src/foo.c
        cc -o lib_foo.o foo.o -bE:foo_add.exp -bI:bar_add.exp -bM:SRE -bnoentry
        cc -c ./src/bar.c
        cc -o lib_bar.o bar.o -bE:bar_add.exp -bI:foo_add.exp -bM:SRE -bnoentry
        ar rv lib_foobar.a lib_foo.o
ar: Creating an archive file lib_foobar.a.
a - lib_foo.o
        ar rv lib_foobar.a lib_bar.o
a - lib_bar.o
$ ls -l *.a
-rw-r--r--   1 jasper   usr           3757 Mar 05 13:28 lib_foobar.a
$
```

Figure 44. Using make rules from Solaris with the AIX 5L make command

To see how wrong things can go, look at the example, in Figure 45 on page 181, of a makefile being evaluated using the makerules from HP-UX11. The .mk file from HP-UX has been altered in the following way:

- Removed the $ = $ rule
- Removed commands: lines
- Removed several empty single suffix inference rules
- Corrected the .SUFFIX line

In Figure 45, you can see what happens when we try to use the makerules from HP-UX to compile your example.

```
$ make clean
        rm -f *.o
        rm -f *.a
        rm -f foobar
$ make lib_foobar.a
        cc -c ./src/foo.c
        cc -o lib_foo.o foo.o -bE:foo_add.exp -bI:bar_add.exp -bM:SRE -bnoentry
        cc -c ./src/bar.c
        cc -o lib_bar.o bar.o -bE:bar_add.exp -bI:foo_add.exp -bM:SRE -bnoentry
        ar rv lib_foobar.a lib_foo.o
ar: Creating an archive file lib_foobar.a.
a - lib_foo.o
        ar rv lib_foobar.a lib_bar.o
a - lib_bar.o
$ make MAKERULES=/usr/ccs/lib/hpux11e.mk clean
        rm -f *.o
        rm -f *.a
        rm -f foobar
$ make -k MAKERULES=./usr/ccs/lib/hpux11e.txt lib_foobar.a
        cc -c /foo.c
cc: 1501-228 input file /foo.c not found
make: 1254-004 The error code from the last command is 252.
 (continuing)
        cc -c /bar.c
cc: 1501-228 input file /bar.c not found
make: 1254-004 The error code from the last command is 252.
 (continuing)
Target "lib_foobar.a" did not make because of errors.
$
```

Figure 45. Trying to use make rules from HP-UX with AIX 5L make

So if you import the makerules from another `make` command, you have to be very careful and do extensive testing to be sure that it actually works.

Also bear in mind that you have not really ported your makefiles to use the `make` command under AIX 5L, but have taken the features from your target platforms `make` command with you to AIX 5L. If you port the makefiles to another platform, you will have to go through the same procedure again.

6.2.4 Exit values from the make command

If you are using the exit values from the `make` command, you have to be careful. The POSIX standard states:

“When the -q option is specified, the make utility shall exit with one of the following values:

0	Successful completion.
1	The target was not up-to-date.
>1	An error occurred.

When the -q option is not specified, the make utility shall exit with one of the following values:

0	Successful completion.
>0	An error occurred.”

If you are using scripts that rely on the exit value from the `make` command, you should check those scripts, and do some testing to make sure that the script still works.

Chapter 7. System functions

System functions cover a large number of interfaces. To keep this redbook within a manageable scope, we have focused on the most commonly used facilities. Coding examples are provided to give assistance and guidance where needed.

7.1 Priority manipulation

If the application being ported uses priority levels to configure the run-time environment, AIX 5L provides several ways of manipulating priorities. The priority definition within AIX 5L is similar to other UNIX-based operating systems, and has the following characteristics:

```
PRIORITY_MIN = 0, PRIORITY_MAX = 255
```

The lower the number, the more favored the process.

A priority may be defined within the bounds of:

PRIORITY_MIN < **priority** < PRIORITY_MAX

Note

Some implementations use the opposite priority layout where the higher the number, the more favored the process. When using priority values to control behavior, check that your code is using the correct model.

AIX 5L uses predefined priorities for system processes to ensure correct operation. Several system defaults are listed in Table 58.

Table 58. Example of AIX 5L default priorities

System activity	Priority
Swapper	0
Scheduler	16
IPC Messages	27
User process	60 (base level 40 + default nice 20)

Important

Assigning an application a priority more favorable than system processes may cause system hangs which can only be resolved by a system reset.

Table 59 shows the commonly used mechanisms for priority setup and manipulation. AIX 5L, Solaris 8, HP-UX 11 and Tru64 5.1 all support the System V/BSD getpriority()/setpriority() subroutines listed. Information on AIX 5L priority handling can be found in *Performance Management Guide - Performance Overview of the CPU Scheduler*, which can be found in the AIX 5L online documentation. Refer to Section 4.2, “Online documentation” on page 83 for more information.

Table 59. Priority manipulation subroutines

Routine	**Purpose**
getpri	Determines the scheduling priority of a running process.
setpri	Changes the priority of a running process to a fixed priority.
getpriority	Determines the nice value of a running process.
nice	Increments the nice value of the current process.
setpriority	Sets the nice value of a running process.

Multithreaded applications may define their own internal scheduling and priority environment. This is discussed in Chapter 10, “POSIX threads” on page 307. When looking at application processes, AIX 5L has two distinct types of priority manipulation routines. Both manipulate process priorities, but getpri() and setpri() are only concerned with the base scheduling priority. If a priority is set with setpri(), that priority is fixed and will not be adjusted by the operating system. Using fixed priorities allows a scheduling hierarchy to be explicitly defined that will not be subject to automatic process priority adjustment.

Note

Only processes that have root user authority can set a process scheduling priority to a fixed value. If the process has multiple threads, all threads have their priority reset and the scheduling policy is changed to SCHED_RR. For more information, please see Chapter 10, “POSIX threads” on page 307.

The routines getpriority(), nice(), and setpriority() manipulate the nice value of the process but do not provide fixed priority behavior. The following example code segment shows the start of the priority1.c program that gets and sets priority values:

```
#include <stdio.h>
#include <sys/sched.h>

/* priority for important task */
#define FAVORED PRIORITY_MIN+1

/* priority for non-important task */
#define BACKGROUND PRIORITY_MAX-1

main()
{
    int pval    = -1;   /* Define and initialize */
    int opval   = -1;   /* Define and initialize */
    const pid_t pid = 0;    /* Define and initialize */

    /* Output priority limits */
    printf("PRIORITY_MIN = %d\n",PRIORITY_MIN);
    printf("PRIORITY_MAX = %d\n",PRIORITY_MAX);

    /* Get initial priority */
    if((opval=getpri(pid))==-1) {
        perror("getpri #1 >   ");
        exit(1);
    }

    printf("Default priority = %d\n",opval);

    /* Set priority to favored */
    if((pval=setpri(pid,FAVORED))==-1) {
        perror("setpri #1 >  ");
        exit(1);
    }
    .
    .
    .
```

The output of the full program is:

```
PRIORITY_MIN = 0
PRIORITY_MAX = 255
Default priority = 60
After change, current priority = FAVORED = 1
After change, current priority = BACKGROUND = 254
Reset back to original priority = 60
```

Workload Manager is supplied as part of AIX 5L and provides the facilities to dedicate processing resources to particular applications. See Section 4.16.2, “Limiting resource usage with WLM” on page 115. As WLM may be used with little or no source code modification, it can provide an easily tuned and flexible run time environment.

7.2 CPU manipulation

For some processing environments, it may be necessary to override the natural load balancing behavior of AIX 5L and explicitly bind processes to physical processors. AIX 5L provides a command line and API to both bind and unbind all kernel threads in a process.

The `bindprocessor` command can report the number of CPUs configured and perform bind or unbind operations.

The bindprocessor subroutine can only perform bind or unbind operations with no resource reporting mechanism. The calling process must have the root user authority to use this subroutine. The POSIX style sysconf subroutine may be used to gather information about system resources including the number of CPUs and so forth. Both subroutines are discussed in *General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation.

The program cpu_bind.c checks for the number of CPUs, binds to each CPU in turn, and then unbinds itself. A `ps` command is used to show if threads are bound to any CPU resource. The information appears in the BND column.

```
#include <unistd.h>     /* needed by 'sysconf' */
#include <sys/processor.h>  /* needed by 'bindprocessor' */
#include <sys/types.h>      /* needed by 'getpid' */
#include <sys/errno.h>      /* needed by 'getpid' */
#include <stdlib.h>     /* needed by 'system' */

long    int liret = -1; /* Define and initialize */
int iret    = -1;       /* Define and initialize */
cpu_t   cpu = -1;       /* Define and initialize */

/* Setup string for 'ps' command */
const char  ps_command[] = "ps -o THREAD";

main()
{
    /* How many CPUs do we have? */
    if ((liret = sysconf(_SC_NPROCESSORS_CONF)) == -1) {
        perror("sysconf #1");
        exit(1);
    }
    printf("%d Processors configured\n", liret);

    /* How many processors are online? */
    if ((liret = sysconf(_SC_NPROCESSORS_ONLN)) == -1) {
        perror("sysconf #2");
        exit(1);
    }

    printf("%d Processors online\n", liret);

    /* Run 'ps' command */
    if(system(ps_command) != 0) {
        perror("system >  ");
        exit(1);
    }

    /* loop through all CPUs */
    for(cpu=0; cpu<liret; cpu++) {

        /* bind process kerner threads to a CPU */
        if(bindprocessor(BINDPROCESS,getpid(),cpu)!=0) {
            perror("bindprocessor on CPU %d",cpu);
            exit(1);
        }

        printf("Bound to logical CPU %d\n",cpu);

        /* Run 'ps' command */
        system(ps_command);
    }

    /* Perform unbind operations */
    if(bindprocessor(BINDPROCESS,getpid(),PROCESSOR_CLASS_ANY)!=0) {
        perror("bindprocessor -unbind");
        exit(1);
    }

    printf("Process has escaped! (unbound)\n");

    /* Run 'ps' command */
    system(ps_command);
}
```

Edited highlights of the output are:

```
2 Processors configured
2 Processors online

USER    PID  PPID  TID ST  CP  PRI  SC   WCHAN       F     TT BND COMMAND
pnutt 16318 15552    - A    1   60   1       -  200001  pts/2   - cpu_bind

Bound to logical CPU 0
USER    PID  PPID  TID ST  CP  PRI  SC   WCHAN       F     TT BND COMMAND
pnutt 16318 15552    - A    1   60   1       -  200001  pts/2   0 cpu_bind

Bound to logical CPU 1

USER    PID  PPID  TID ST  CP  PRI  SC   WCHAN       F     TT BND COMMAND
pnutt 16318 15552    - A    1   60   1       -  200001  pts/2   1 cpu_bind
```

```
Process has escaped! (unbound)

USER     PID  PPID  TID ST  CP  PRI  SC   WCHAN        F     TT BND COMMAND
pnutt 16318 15552     - A    1   60   1       -   200001  pts/2   - cpu_bind
```

7.3 Memory locking/pinning

To deliver improved determinism, developers may use memory locking to guard against unwanted paging of an applications text and data regions. AIX 5L provides the System V style plock subroutine for locking and unlocking of application address space in physical memory. The calling process must have the root user authority to use this subroutine. The following example code increases stack space (required before a plock) and then locks and unlocks the text and data regions:

```
#include <stdio.h>  /* for printf */
#include <ulimit.h> /* for ulimit */
#include <sys/lock.h>   /* for plock */

main ()
{
    long    int liret   = -1;
    off_t   newlim  = -1;

    /* Get the lowest valid stack address */
    if((liret=ulimit(GET_STACKLIM,0)) == -1) {
        perror("ulimit#1 >  ");
        exit(1);
    }

    printf("Lowest valid stack address is %x\n",liret);

    /* Get 8 pages more stack space */
    newlim  = liret- (PAGESIZE*8);

    /* Set the lowest valid stack address */
    if((liret=ulimit(SET_STACKLIM,newlim)) == -1) {
        perror("ulimit#2 >  ");
        exit(1);
    }

    printf("Lowest valid stack address is now %x\n",liret);

    /* lock memory */
    if(plock(PROCLOCK)!=0) {
        perror("plock #1 >  ");
        exit(1);
    }

    printf("Process text and data regions now locked\n");

    /* unlock memory */
    if(plock(UNLOCK)!=0) {
        perror("plock #2 >  ");
        exit(1);
    }
    printf("Process text and data regions now unlocked\n");
}
```

The following output is produced after the mem_lock.c application is run:

```
Lowest valid stack address is 2df23000
Lowest valid stack address is now 2df1b000
Process text and data regions now locked
Process text and data regions now unlocked
```

For further information, please refer to *Performance Management Guide - Code-Optimization Techniques and General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation.

7.4 How to determine system configuration

Applications that can run on different target systems sometimes need to get information about their run time environment. This data can be used to ensure the best use of available resources. AIX 5L provides the `lsattr` command and POSIX style subroutine sysconf. Both are described in Table 60.

Table 60. AIX 5L system configuration determination

Name	Function
sysconf	Determines the current value of a specified system limit or option.
lsattr	Displays attribute characteristics and possible values of attributes for devices in the system.

Note

Although sysconf is defined within the POSIX standard, individual systems may have different name definitions. If the Name parameter is invalid, a value of -1 is returned, and the errno global variable is set to indicate the error. If the Name parameter is valid but is a variable not supported by the system, a value of -1 is returned, and the errno global variable is set to a value of EINVAL.

7.5 Shared or mapped memory

AIX 5L supports the System V style of shared and mapped memory functions, as shown in Table 61 on page 190. Solaris 8, HP-UX 11, and Tru64 5.1

support the subroutines listed apart from mincore which does not appear in HP-UX 11 and Tru64 5.1.

Table 61. AIX 5L shared or mapped memory subroutines

Name	Function
shmat	Attaches a shared memory segment or a mapped file to the current process.
shmctl	Controls shared memory operations.
shmget	Gets shared memory segments.
shmdt	Detaches a shared memory segment.
ftok	Generates a standard interprocess communication key.
mmap	Maps a file system object into virtual memory.
madvise	Advises the system of expected paging behavior.
mprotect	Modifies access protections for memory mapping.
msync	Synchronizes a mapped file.
munmap	Unmaps a mapped region.
mincore	Determines residency of memory pages.

If more than 11 shared memory segments are used in a 32-bit application, AIX 5L requires an environment variable to be set to ensure correct operation:

```
EXTSHM=ON
```

If an application creates and attaches 12 regions and fails on the twelfth because the EXTSHM variable was not set (causing error: too many open files), it is necessary to remove the existing segments, using the commands `ipcs -m` and `ipcrm -m`. Once the segments have been removed, the environment variable may be set and the program rerun.

Note

If a single shared memory region larger than 256 MB is used, the system automatically works as if the EXTSHM environment variable is set, even if it was not defined explicitly.

When this functionality is enabled, the attached shared memory regions are tracked using an extra level of indirection rather than consuming a segment register for each attached region.

For more information, please see *General Programming Concepts*, which can be found in the AIX 5L online documentation, or the IBM Redbook, *C and C++ Application Development on AIX,* SG24-5674.

The following example, shared_mem.c, creates a shared memory region, uses `fork` to create a number of children, and then communicates through a structure held in the shared memory. The parent communicates with all children and requests them to exit, removes the shared memory region and then exits.

```
#include <stdio.h>  /* Needed for printf */
#include <sys/shm.h>    /* Needed for shmget */
#include <sys/types.h>  /* Needed for fork */
#include <unistd.h> /* Needed for fork */

/* protections for shmget */
#define RUSER   0444
#define WUSER   0222

/* control->status definitions */
#define PRESENT 0xbeef
#define DEAD    0xdead
#define STATUS  0xdeaf
#define OK  0xfeed
#define QUIT    0xfade

#define NKIDS   4   /* How many children? */

/* Setup the control structure */
struct control_t {
    int             kid_number;
    int             status;
    int             resp;
} *control;

int identity    = -1;           /* Define and initialize */
int ix  = -1;               /* Define and initialize */

main()
{

    pid_t           pid = -1;       /* Define and initialize */
    int             nkids = -1;     /* Define and initialize */
    int             child = 0;      /* Set child to false */
    int             parent = 0;     /* Set parent to false */
    const   key_t   mkey    = 0xf00;    /* Define and initialize */
    const   int shmsize = PAGESIZE; /* Define and initialize */
    int shmid   = -1;           /* Define and initialize */

    /* Get some shared memory for the control structure */
    /* First create and get a shared memory structure */
    if((shmid=shmget(mkey, shmsize, RUSER | WUSER | IPC_CREAT)) == -1) {
        perror("shmget >  ");
        exit(1);
    }

    /*
    * OK, now attach the shared memory region
    * Attach it to the commumications control structure
    */
    if((int)(control=(struct control_t *)shmat(shmid, 0, 0)) == -1) {
        perror("shmat >  ");
        exit(1);
    }

    printf("Parent >  Making children\n");

    /* Make NKIDS children */
    for (ix = 0; ix < NKIDS; ix++) {
        control->kid_number = -1;
        control->status = DEAD; /* Setup initial value */

        /* begat a child */
        if ((pid = fork()) == -1) {
            printf("fork #1");
            exit(1);
        }

        /* Check to see if child or parent */
        if (pid == 0) {
            /* CHILD */
            child = 1;  /* Set child true */
            parent = 0; /* and parent false -just to make sure */
            identity    = ix;   /* Set 'local' identity */
            control->status = PRESENT;  /* Tell parent I am alive */
            goto CHILD;
        } else {
            /* PARENT */
            parent = 1; /* I am the parent */
            child = 0;  /* not the child */

                        /*
                        * Check if child is alive, wait for the PRESENT flag
                        * wait in a loop but use usleep (microsecond delay) to
                        * check every so often without 'spinning' on the CPU
                        * not suprisingly, 100 000 uSecs gives a 10Hz update
            */
            while (control->status != PRESENT) usleep(100000);
        }


    }

    /* PARENT SECTION */
    /* NKIDS children have been created */

    /* OK, communicate with children, one at a time */
    for (ix = 0; ix < NKIDS; ix++) {
        printf("Parent >  Checking child #%d\n",ix);
        control->status = STATUS;   /* Ask for STATUS */
        control->kid_number = ix;   /* Say which child */
        while (control->status == STATUS) usleep(100000);   /* Wait for response */
        printf("Parent >  Received response from child #%d\n",control->resp);
    }

    /* Have spoken to everyone, tell all children to quit */
    printf("Parent >  Children quit!\n");
```

```
    control->status = QUIT;

    sleep(1);   /* Wait a bit before removing shared mem */

    /* Detach from the shared mem */
    if(shmdt(control) != 0) {
        perror("shmdt >  ");
        exit(1);
    }

    /* Remove the segment from the system */
    if(shmctl(shmid,IPC_RMID,NULL) != 0) {
        perror("shmctl >  ");
        exit(1);
    }

    printf("Parent >  Completed\n");
    exit(0);
CHILD:
    /* CHILD */

    /* Tell everyone I am waiting */
    printf("Child >  #%d waiting\n",identity);

    /* Wait for something to do */
    while(control->status != QUIT) {
        /* Is it me? */
       while (control->kid_number != identity && control->status != QUIT) usleep(100000);

        /* It is me, what do I have to do? */
        if(control->status == STATUS) {
            /* OK, just a status check */
            control->kid_number = -1;   /* Clear */
            control->status = OK;       /* Set OK */
            control->resp   = identity; /* Say who it is */
        }
    }

    /* Have found QUIT, bye bye */
    printf("Child >  #%d quitting\n",identity);

    exit(0);
}
```

An output of the program is:

```
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #3 waiting
Parent >  Checking child #0
Parent >  Received response from child #0
Parent >  Checking child #1
Parent >  Received response from child #1
Parent >  Checking child #2
Parent >  Received response from child #2
Parent >  Checking child #3
Parent >  Received response from child #3
Parent >  Children quit!
Child >  #0 quitting
Child >  #1 quitting
Child >  #2 quitting
Child >  #3 quitting
Parent >  Completed
```

7.6 Signals

If signals are used in exotic ways, there may be portability issues with regard to individual implementation characteristics. AIX 5L provides the POSIX, BSD, and System V style subroutine functions. Given this degree of flexibility, AIX 5L should be able to handle most signal code encountered during a port. If work is required, we would recommend moving to the POSIX implementation for improved onward portability. If the application exhibits incorrect behavior, there may be differences in subroutine parameter values or default behavior on signal delivery. In this case, consult the man pages and check for hard coded values. For further information, please consult *General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation. Table 62 lists the commonly used signals for AIX 5L and other major UNIX platforms.

Table 62. Signals

AIX 5L	**Tru64 v5.1 or higher**	**Sun Solaris 8**	**HP-UX 11i**
SIGABRT	SIGABRT	SIGABRT	SIGABRT
	SIGAIO		
SIGALRM	SIGALRM	SIGALRM	SIGALRM
SIGALRM1			
SIGBUS	SIGBUS	SIGBUS	SIGBUS
		SIGCANCEL	
SIGCHLD	SIGCHLD	SIGCHLD	SIGCHLD
SIGCONT	SIGCONT	SIGCONT	SIGCONT
SIGCPUFAIL			
SIGDANGER			
SIGEMT	SIGEMT	SIGEMT	
SIGFPE	SIGFPE	SIGFPE	SIGFPE
		SIGFREEZE	
SIGGRANT			
SIGHUP	SIGHUP	SIGHUP	SIGHUP
SIGILL		SIGILL	SIGILL
	SIGINFO		
SIGINT	SIGINT	SIGINT	SIGINT
SIGIO	SIGIO	SIGIO	
	SIGIOINT		

AIX 5L	Tru64 v5.1 or higher	Sun Solaris 8	HP-UX 11i
	SIGIOT		
SIGKAP			
SIGKILL	SIGKILL	SIGKILL	SIGKILL
	SIGLOST		
		SIGLWP	
SIGMIGRATE			
SIGMSG			
SIGPIPE	SIGPIPE	SIGPIPE	SIGPIPE
	SIGPOLL	SIGPOLL	SIGPOLL
SIGPRE			
SIGPROF	SIGPROF	SIGPROF	SIGPROF
	SIGPTY		
SIGPWR	SIGPWR	SIGPWR	
SIGQUIT	SIGQUIT	SIGQUIT	SIGQUIT
	SIGRESV		
SIGRETRACT			
		SIGRTMAX	SIGRTMAX
		SIGRTMIN	SIGRTMIN
SIGSAK			
SIGSEGV	SIGSEGV	SIGSEGV	SIGSEGV
SIGSOUND			
SIGSTOP	SIGSTOP	SIGSTOP	SIGSTOP
SIGSYS	SIGSYS	SIGSYS	SIGSYS
SIGTERM	SIGTERM	SIGTERM	SIGTERM
		SIGTHAW	
SIGTRAP	SIGTRAP	SIGTRAP	SIGTRAP
SIGTSTP	SIGTSTP	SIGTSTP	SIGTSTP
SIGTTIN	SIGTTIN	SIGTTIN	SIGTTIN
SIGTTOU	SIGTTOU	SIGTTOU	SIGTTOU
SIGURG	SIGURG	SIGURG	SIGURG
SIGUSR1	SIGUSR1	SIGUSR1	SIGUSR1
SIGUSR2	SIGUSR2	SIGUSR2	SIGUSR2
SIGVIRT			
SIGVTALRM	SIGVTALRM	SIGVTALRM	SIGVTALRM
SIGWAITING		SIGWAITING	
SIGWINCH	SIGWINCH	SIGWINCH	
SIGXCPU	SIGXCPU	SIGXCPU	SIGXCPU
SIGXFSZ	SIGXFSZ	SIGXFSZ	SIGXFSZ

There are many different ways of generating and manipulating signals under UNIX. The main standard interfaces for signal setup and manipulation are listed in Table 63.

Table 63. Standard signal functions

Process signal and mask functions	Description
kill, killpg, raise, alarm	Sends a signal to an executing program.
sigaction, sigvec, signal, siginterrupt	Specifies the action to take upon delivery of a signal.
sigemptyset, sigfillset, sigaddset, sigdelset, sigismember	Creates and manipulates signal masks.
sigpending	Determines the set of signals that are blocked from delivery.
sigprocmask, sigsetmask, sigblock	Sets signal masks.
sigset, sighold, sigrelse, sigignore, sigpause	Enhances the signal facility and provides signal management.
sigsetjmp, siglongjmp	Saves and restores stack context and signal masks.
sigstack	Sets signal stack context.
sigsuspend	Changes the set of blocked signals.
ssignal, gsignal	Implements a software signal facility.
pthread_kill	Sends a signal to a thread.

AIX 5L, Solaris 8, HP-UX 11, and Tru64 5.1 all support the functions listed but with one difference: HP-UX uses sigvector in place of sigvec, but with the same syntax. In theory, this should only affect legacy code using BSD style calls.

If the source system has a signals implementation that does not match the standards available using AIX 5L, look at the code examples that follow. The code is an extension of the shared memory program and takes advantage of signals to communicate with the child processes. The first example, signals1.c, uses the System V style implementation, and the second example, signals2.c, uses the POSIX style. Both programs generate child processes and initiate specific behavior by sending signals. The response of the signal handler my_handler is seen from the program output and shared memory structure counting.

```
/* signals1.c */
#include <stdio.h>  /* Needed for printf */
#include <sys/shm.h>    /* Needed for shmget */
#include <sys/types.h>  /* Needed for fork */
#include <unistd.h>     /* Needed for fork, getppid and getpgrp */
#include <signal.h> /* Needed for sighold */

/* protections for shmget */
#define RUSER   0444
#define WUSER   0222

/* control->status definitions */
#define PRESENT 0xbeef
#define DEAD    0xdead
#define STATUS  0xdeaf
#define OK  0xfeed
#define QUIT    0xfade

#define NKIDS   4   /* How many children? */

/* Setup the control structure */
struct control_t {
    int             kid_number;
    int             status;
    int             resp;
} *control;

int identity    = -1;           /* Define and initialize */
int ix  = -1;               /* Define and initialize */
pid_t   pgid    = -1;           /* Define and initialize */
long    *old_handler    = (long *)-1;   /* Define and initialize */

main()
{
    pid_t           pid = -1;       /* Define and initialize */
    int             nkids = -1;     /* Define and initialize */
    int             child = 0;      /* Set child to false */
    int             parent = 0;     /* Set parent to false */
    key_t   mkey    = 0xf00;    /* Define and initialize */
    const   int shmsize = PAGESIZE; /* Define and initialize */
    int shmid   = -1;           /* Define and initialize */
    extern void my_handler();       /* Define handler */


    /* Get some shared memory for the control structure */
    /* First create and get a shared memory structure */
    if((shmid=shmget(mkey, shmsize, RUSER | WUSER | IPC_CREAT)) == -1) {
        perror("shmget >  ");
        exit(1);
    }

    /*
    * OK, now attach the shared memory region
    * Attach it to the commumications control structure
    */
    if((int)(control=(struct control_t *)shmat(shmid, 0, 0)) == -1) {
        perror("shmat >  ");
        exit(1);
    }

    /* Get process group id */
    pgid=getpgrp();
```

```
/* Setup signal handler */
if((old_handler=(long *)signal(SIGUSR1,&my_handler)) == (long *)-1) {
    perror("signal#1 >  ");
    exit(1);
}
/* Setup signal handler */
if((old_handler=(long *)signal(SIGUSR2,&my_handler)) == (long *)-1) {
    perror("signal#2 >  ");
    exit(1);
}
/* Setup signal handler */
if((old_handler=(long *)signal(SIGINT,&my_handler)) == (long *)-1) {
    perror("signal#3 >  ");
    exit(1);
}

printf("Parent >  Making children\n");

/* Make NKIDS children */
for (ix = 0; ix < NKIDS; ix++) {
    control->kid_number = -1;
    control->status = DEAD; /* Setup initial value */

    /* begat a child */
    if ((pid = fork()) == -1) {
        printf("fork #1");
        exit(1);
    }

    /* Check to see if child or parent */
    if (pid == 0) {
        /* CHILD */
        child = 1;  /* Set child true */
        parent = 0; /* and parent false -just to make sure */
        identity    = ix;   /* Set 'local' identity */
        control->status = PRESENT;  /* Tell parent I am alive */
        goto CHILD;
    } else {
        /* PARENT */
        parent = 1; /* I am the parent */
        child = 0;  /* not the child */

                    /*
                    * Check if child is alive, wait for the PRESENT flag
                    * wait in a loop but use usleep (microsecond delay) to
                    * check every so often without 'spinning' on the CPU
                    * not suprisingly, 100 000 uSecs gives a 10Hz update
        */
        while (control->status != PRESENT) usleep(100000);
    }


}

/* PARENT SECTION */
/* NKIDS children have been created */

/*
* OK, communicate with children by sending a signal
* first, make sure I do not get it
*/
sighold(SIGUSR1);
sighold(SIGUSR2);
```

```
    sighold(SIGINT);

    /* Clear the comms */
    control->resp   = 0;

    /* Send the signal */
    if(killpg(pgid,SIGUSR1) != 0) {
        perror("killpg#1 >  "); exit(1);
    }

    /* Send the signal */
    if(killpg(pgid,SIGUSR2) != 0) {
        perror("killpg#2 >  "); exit(1);
    }

    /* Send the signal */
    if(killpg(pgid,SIGINT) != 0) {
        perror("killpg#3 >  "); exit(1);
    }

    /* Check all children have reported in */
    if(control->resp != NKIDS) {
        printf("\nparent >  We have a problem Houston!\n");
        printf("parent >  only %d children responded!\n\n",control->resp);
    } else {
        printf("parent >  all children responded!\n");
    }

    /* Have spoken to everyone, tell all children to quit */
    printf("Parent >  Children quit!\n");
    control->status = QUIT;

    sleep(1);   /* Wait a bit before removing shared mem */

    /* Detach from the shared mem */
    if(shmdt(control) != 0) {
        perror("shmdt >  ");
        exit(1);
    }

    /* Remove the segment from the system */
    if(shmctl(shmid,IPC_RMID,NULL) != 0) {
        perror("shmctl >  ");
        exit(1);
    }

    printf("Parent >  Completed\n");
    exit(0);
CHILD:
    /* CHILD */

    /* Tell everyone I am waiting */
    printf("Child >  #%d waiting\n",identity);

    /* Wait for something to do */
    while(control->status != QUIT) {
        /* Is it me? */
       while (control->kid_number != identity && control->status != QUIT) usleep(100000);

        /* It is me, what do I have to do? */
        if(control->status == STATUS) {
            /* OK, just a status check */
            control->kid_number = -1;   /* Clear */
            control->status = OK;       /* Set OK */
            control->resp   = identity; /* Say who it is */
        }
    }

    /* Have found QUIT, bye bye */
    printf("Child >  #%d quitting\n",identity);

    exit(0);
}
void my_handler(signal, code, scp)
int             signal;
int             code;
struct sigcontext *scp;
{
    /* Have caught a signal -work on it */
    switch(signal) {
    case SIGUSR1:
        /* Increment the shared memory counter */
        control->resp++;
        break;
    case SIGUSR2:
        /* Say hello */
        printf("Child#%d says 'hello world'\n",identity);
        break;
    default:
        /* Where did that come from? */
        printf("Child#%d >  Unexpected signal -foo!\n",identity);
    }
    return;
}
```

An example output from signals1.c is:

```
Parent >  Making children
Child >  #2 waiting
Child#2 says 'hello world'
Child#2 >  Unexpected signal -foo!
Child >  #2 quitting
Parent >  Making children
Child >  #0 waiting
Child#0 says 'hello world'
Child#0 >  Unexpected signal -foo!
Child >  #0 quitting
Parent >  Making children

parent >  We have a problem Houston!
parent >  only 1 children responded!

Parent >  Children quit!
Parent >  Completed
```

Although the program is functioning correctly, the output is displayed out of order due to the asynchronous processing of the signal handler and printf. The “only one child responded” message is misleading, as the parent program checks the shared memory value before all the children have reported. So the code is running OK, but reporting an error because of the timing.

The program signals2.c uses POSIX style signals and is listed below:

```
/* signals2.c */
#include <stdio.h>  /* Needed for printf */
#include <sys/shm.h>    /* Needed for shmget */
#include <sys/types.h>  /* Needed for fork */
#include <unistd.h>     /* Needed for fork, getppid and getpgrp */
#include <signal.h> /* Needed for sighold etc */

/* protections for shmget */
#define RUSER   0444
#define WUSER   0222

/* control->status definitions */
#define PRESENT 0xbeef
#define DEAD    0xdead
#define STATUS  0xdeaf
#define OK  0xfeed
#define QUIT    0xfade

#define NKIDS   4   /* How many children? */

/* Setup the control structure */
struct control_t {
    int             kid_number;
    int             status;
    int             resp;
} *control;

int identity    = -1;           /* Define and initialize */
int ix  = -1;               /* Define and initialize */
pid_t   pgid    = -1;           /* Define and initialize */
long    *old_handler    = (long *)-1;   /* Define and initialize */
sigset_t    set, old_set;       /* Define */
struct sigaction action, oaction;


main()
{
    pid_t           pid = -1;       /* Define and initialize */
    int             nkids = -1;     /* Define and initialize */
    int             child = 0;      /* Set child to false */
    int             parent = 0;     /* Set parent to false */
    key_t   mkey    = 0xf00;    /* Define and initialize */
    const   int shmsize = PAGESIZE; /* Define and initialize */
    int shmid   = -1;           /* Define and initialize */
    extern void my_handler();       /* Define handler */


    /* Get some shared memory for the control structure */
    /* First create and get a shared memory structure */
    if((shmid=shmget(mkey, shmsize, RUSER | WUSER | IPC_CREAT)) == -1) {
        perror("shmget >  "); exit(1); }

        /*
        * OK, now attach the shared memory region
        * Attach it to the commumications control structure
    */
    if((int)(control=(struct control_t *)shmat(shmid, 0, 0)) == -1) {
        perror("shmat >  "); exit(1); }

    /* Get process group id */
    pgid=getpgrp();
```

```
sigemptyset(&set);      /* Get a clean set */
sigaddset(&set,SIGUSR1);    /* Add SIGUSR1 */
sigaddset(&set,SIGUSR2);    /* Add SIGUSR2 */
sigaddset(&set,SIGINT);     /* Add SIGINT */

/* Setup the handler */
action.sa_handler = &my_handler;

/* Block other signals whilst in handler */
action.sa_mask  = set;

/* Setup signal handler */
if (sigaction(SIGUSR1, &action, &oaction) != 0) {
    perror("sigaction #1 >  "); exit(1); }

/* Setup signal handler */
if (sigaction(SIGUSR2, &action, &oaction) != 0) {
    perror("sigaction #2 >  "); exit(1); }

/* Setup signal handler */
if (sigaction(SIGINT, &action, &oaction) != 0) {
    perror("sigaction #3 >  "); exit(1); }

printf("Parent >  Making children\n");

/* Make NKIDS children */
for (ix = 0; ix < NKIDS; ix++) {
    control->kid_number = -1;
    control->status = DEAD; /* Setup initial value */

    /* begat a child */
    if ((pid = fork()) == -1) {
        printf("fork #1"); exit(1); }

    /* Check to see if child or parent */
    if (pid == 0) {
        /* CHILD */
        child = 1;  /* Set child true */
        parent = 0; /* and parent false -just to make sure */
        identity    = ix;   /* Set 'local' identity */
        control->status = PRESENT;  /* Tell parent I am alive */
        goto CHILD;
    } else {
        /* PARENT */
        parent = 1; /* I am the parent */
        child = 0;  /* not the child */

                    /*
                    * Check if child is alive, wait for the PRESENT flag
                    * wait in a loop but use usleep (microsecond delay) to
                    * check every so often without 'spinning' on the CPU
                    * not suprisingly, 100 000 uSecs gives a 10Hz update
        */
        while (control->status != PRESENT) usleep(100000);
    }
}

/* PARENT SECTION */
/* NKIDS children have been created */

/*
* OK, communicate with children by sending a signal
```

```
    * first, make sure I do not get it
    */

    /* Do the 'block' */
    if(sigprocmask(SIG_BLOCK,&set,&old_set) == 1) {
        perror("sigprocmask#1 >  "); exit(1); }

    /* Clear the comms */
    control->resp   = 0;

    /* Send the signal */
    if(killpg(pgid,SIGUSR1) != 0) {
        perror("killpg#1 >  "); exit(1); }

    /* Send the signal */
    if(killpg(pgid,SIGUSR2) != 0) {
        perror("killpg#2 >  "); exit(1); }

    /* Send the signal */
    if(killpg(pgid,SIGINT) != 0) {
        perror("killpg#3 >  "); exit(1); }

    /* Check all children have reported in */
    if(control->resp != NKIDS) {
        printf("\nparent >  We have a problem Houston!\n");
        printf("parent >  only %d children responded!\n\n",control->resp);
    } else {
        printf("parent >  all children responded!\n");
    }

    /* Have spoken to everyone, tell all children to quit */
    printf("Parent >  Children quit!\n");
    control->status = QUIT;

    sleep(1);   /* Wait a bit before removing shared mem */

    /* Detach from the shared mem */
    if(shmdt(control) != 0) {
        perror("shmdt >  "); exit(1); }

    /* Remove the segment from the system */
    if(shmctl(shmid,IPC_RMID,NULL) != 0) {
        perror("shmctl >  "); exit(1); }

    printf("Parent >  Completed\n");

    exit(0);
CHILD:
    /* CHILD */

    /* Tell everyone I am waiting */
    printf("Child >  #%d waiting\n",identity);

    /* Wait for something to do */
    while(control->status != QUIT) {
        /* Is it me? */
       while (control->kid_number != identity && control->status != QUIT) usleep(100000);

        /* It is me, what do I have to do? */
        if(control->status == STATUS) {
            /* OK, just a status check */
            control->kid_number = -1;   /* Clear */
            control->status = OK;       /* Set OK */
            control->resp   = identity; /* Say who it is */
        }
    }

    /* Have found QUIT, bye bye */
    printf("Child >  #%d quitting\n",identity);

    exit(0);
}
void my_handler(signal, code, scp)
int             signal;
int             code;
struct sigcontext *scp;
{
    /* Have caught a signal -work on it */
    switch(signal) {
    case SIGUSR1:
        /* Increment the shared memory counter */
        control->resp++; break;
    case SIGUSR2:
        /* Say hello */
        printf("Child#%d says 'hello world'\n",identity); break;
    default:
        /* Where did that come from? */
        printf("Child#%d >  Unexpected signal -foo!\n",identity);
    }
    return;
}
```

An example output of signals2 is:

```
Parent >  Making children
Child >  #2 waiting
Child#2 >  Unexpected signal -foo!
Child#2 says 'hello world'
Child >  #2 quitting
Parent >  Making children
Child >  #0 waiting
Child#0 >  Unexpected signal -foo!
Child#0 says 'hello world'
Child >  #0 quitting
Parent >  Making children
Child >  #3 waiting
Child#3 says 'hello world'
Child#3 >  Unexpected signal -foo!
Child >  #3 quitting
Parent >  Making children

parent >  We have a problem Houston!
parent >  only 1 children responded!

Parent >  Children quit!
Parent >  Completed
```

In common with signals1, the order of the output and reported error concerning the number of children can be ignored.

Note

Using signals in a threaded process environment can be more complex than the simple examples shown here. Please refer to Chapter 10, “POSIX threads” on page 307 and *General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation.

7.7 Threads

AIX 5L supports a standards compliant threaded environment and offers a highly flexible set of interfaces. To ensure AIX 5L threads are covered in enough detail, there is a separate chapter dedicated to this topic. Please turn to Chapter 10, “POSIX threads” on page 307 for more information.

7.8 Semaphores

AIX 5L provides two styles of semaphore operation: System V and AES. Table 64 lists the available subroutines. For more information, please refer to G*eneral Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation.

Table 64. Semaphore subroutines

Semaphore subroutines	**Description**
semget	Gets a set of semaphores.
semop	Performs semaphore operations.
semctl	Controls semaphore operations.
msem_init	Initializes a semaphore.
msem_lock	Locks a semaphore.
msem_unlock	Unlocks a semaphore.
msem_remove	Removes a semaphore.
msleep	Puts a process to sleep when a semaphore is busy.
mwakeup	Wakes up a process that is waiting on a semaphore.
disclaim	Disclaims the content of a memory address range.

The semaphore1.c program listed below is based upon signals2.c, but uses System V semaphores to coordinate individual phases of execution. Between each signal, there is a semaphore wait to pause execution until the signal handlers or other processing has finished.

```
/* semaphore1.c */
#include <stdio.h>  /* Needed for printf */
#include <sys/shm.h>    /* Needed for shmget */
#include <sys/types.h>  /* Needed for fork */
#include <unistd.h>     /* Needed for fork, getppid and getpgrp */
#include <signal.h> /* Needed for sighold etc */
#include <sys/sem.h>    /* Needed for semget... */

/* protections for shmget */
#define RUSER   0444
#define WUSER   0222

/* control->status definitions */
#define PRESENT 0xbeef
#define DEAD    0xdead
#define STATUS  0xdeaf
#define OK  0xfeed
#define QUIT    0xfade

/* maximum number of semaphore operations */
#define MAX_NUM_SEMS    2

#define NKIDS   4   /* How many children? */

/* Setup the control structure */
struct control_t {
    int             kid_number;
    int             status;
    int             resp;
} *control;

int identity    = -1;           /* Define and initialize */
int ix  = -1;               /* Define and initialize */
pid_t   pgid    = -1;           /* Define and initialize */
long    *old_handler    = (long *)-1;   /* Define and initialize */
sigset_t    set, old_set;       /* Define */
struct sigaction action, oaction;

struct  sembuf sembuf_lock[MAX_NUM_SEMS];
struct  sembuf sembuf_unlock[MAX_NUM_SEMS];

int semid   = -1;


main()
{
    pid_t           pid = -1;       /* Define and initialize */
    int             nkids = -1;     /* Define and initialize */
    int             child = 0;      /* Set child to false */
    int             parent = 0;     /* Set parent to false */
    key_t   mkey    = 0xf00;    /* Define and initialize */
    const   int shmsize = PAGESIZE; /* Define and initialize */
    int shmid   = -1;           /* Define and initialize */
    extern void my_handler();       /* Define handler */
    int itemp   = -1;

    /* Initialise a lock -create a semaphore */
```

```
itemp   = IPC_CREAT|S_IRUSR|S_IWUSR;
if((semid=semget(IPC_PRIVATE,MAX_NUM_SEMS,itemp))==-1) {
    perror("semget #1"); exit(1); }

/* Setup the 'check and set' semaphore */
sembuf_lock[0].sem_num  = 0;
sembuf_lock[0].sem_op   = 0;
sembuf_lock[1].sem_num  = 0;
sembuf_lock[1].sem_op   = NKIDS;

/* Setup the semaphore with a value of NKIDS */
if(semop(semid,sembuf_lock,(size_t)MAX_NUM_SEMS)!=0) {
    perror("semop #1"); exit(1); }

/* Setup the 'decrement' semaphore */
sembuf_unlock[0].sem_num    = 0;
sembuf_unlock[0].sem_op = -1;

/* Get some shared memory for the control structure */
/* First create and get a shared memory structure */
if((shmid=shmget(mkey, shmsize, RUSER | WUSER | IPC_CREAT)) == -1) {
    perror("shmget >  "); exit(1); }

    /*
    * OK, now attach the shared memory region
    * Attach it to the commumications control structure
*/
if((int)(control=(struct control_t *)shmat(shmid, 0, 0)) == -1) {
    perror("shmat >  "); exit(1); }

/* Get process group id */
pgid=getpgrp();

sigemptyset(&set);      /* Get a clean set */
sigaddset(&set,SIGUSR1);    /* Add SIGUSR1 */
sigaddset(&set,SIGUSR2);    /* Add SIGUSR2 */
sigaddset(&set,SIGINT);     /* Add SIGINT */

/* Setup the handler */
action.sa_handler = &my_handler;

/* Block other signals whilst in handler */
action.sa_mask  = set;

/* Setup signal handler */
if (sigaction(SIGUSR1, &action, &oaction) != 0) {
    perror("sigaction #1 >  "); exit(1); }

/* Setup signal handler */
if (sigaction(SIGUSR2, &action, &oaction) != 0) {
    perror("sigaction #2 >  "); exit(1); }

/* Setup signal handler */
if (sigaction(SIGINT, &action, &oaction) != 0) {
    perror("sigaction #3 >  "); exit(1); }

printf("Parent >  Making children\n");

/* Make NKIDS children */
for (ix = 0; ix < NKIDS; ix++) {
    control->kid_number = -1;
    control->status = DEAD; /* Setup initial value */
```

```
        /* begat a child */
        if ((pid = fork()) == -1) {
            printf("fork #1"); exit(1); }

        /* Check to see if child or parent */
        if (pid == 0) {
            /* CHILD */
            child = 1;  /* Set child true */
            parent = 0; /* and parent false -just to make sure */
            identity    = ix;   /* Set 'local' identity */
            control->status = PRESENT;  /* Tell parent I am alive */
            goto CHILD;
        } else {
            /* PARENT */
            parent = 1; /* I am the parent */
            child = 0;  /* not the child */

                        /*
                        * Check if child is alive, wait for the PRESENT flag
                        * wait in a loop but use usleep (microsecond delay) to
                        * check every so often without 'spinning' on the CPU
                        * not suprisingly, 100 000 uSecs gives a 10Hz update
            */
            while (control->status != PRESENT) usleep(100000);
        }
    }

    /* PARENT SECTION */
    /* NKIDS children have been created */

    /*
    * Check the semaphore -only using the first element of the array
    * We will wait here until semval is 0
    */
    if(semop(semid,sembuf_lock,(size_t)1)!=0) {
        perror("semop #2"); exit(1); }

    printf("Parent >  All children alive\n");

    /*
    * OK, communicate with children by sending a signal
    * first, make sure I do not get it
    */

    /* Do the 'block' */
    if(sigprocmask(SIG_BLOCK,&set,&old_set) == 1) {
        perror("sigprocmask#1 >  "); exit(1); }

    /* Clear the comms */
    control->resp   = 0;

    /* Setup the semaphore again */
    if(semop(semid,sembuf_lock,(size_t)MAX_NUM_SEMS)!=0) {
        perror("semop #3"); exit(1); }

    /* Send the signal */
    if(killpg(pgid,SIGUSR1) != 0) {
        perror("killpg#1 >  "); exit(1); }

        /*
        * Check the semaphore -only using the first element of the array
        * We will wait here until semval is 0
    */
```

```
if(semop(semid,sembuf_lock,(size_t)1)!=0) {
    perror("semop #2"); exit(1); }

/* Setup the semaphore again */
if(semop(semid,sembuf_lock,(size_t)MAX_NUM_SEMS)!=0) {
    perror("semop #3"); exit(1); }

/* Send the signal */
if(killpg(pgid,SIGUSR2) != 0) {
    perror("killpg#2 >  "); exit(1); }

    /*
    * Check the semaphore -only using the first element of the array
    * We will wait here until semval is 0
*/
if(semop(semid,sembuf_lock,(size_t)1)!=0) {
    perror("semop #2"); exit(1); }

/* Setup the semaphore again */
if(semop(semid,sembuf_lock,(size_t)MAX_NUM_SEMS)!=0) {
    perror("semop #3"); exit(1); }

/* Send the signal */
if(killpg(pgid,SIGINT) != 0) {
    perror("killpg#3 >  "); exit(1); }

    /*
    * Check the semaphore -only using the first element of the array
    * We will wait here until semval is 0
*/
if(semop(semid,sembuf_lock,(size_t)1)!=0) {
    perror("semop #2"); exit(1); }

/* Check all children have reported in */
if(control->resp != NKIDS) {
    printf("\nparent >  We have a problem Houston!\n");
    printf("parent >  only %d children responded!\n\n",control->resp);
} else {
    printf("parent >  all children responded!\n");
}

/* Setup the semaphore again */
if(semop(semid,sembuf_lock,(size_t)MAX_NUM_SEMS)!=0) {
    perror("semop #3"); exit(1); }

/* Have spoken to everyone, tell all children to quit */
printf("Parent >  Children quit!\n");
control->status = QUIT;

/*
* Check the semaphore -only using the first element of the array
* We will wait here until semval is 0
*/
if(semop(semid,sembuf_lock,(size_t)1)!=0) {
    perror("semop #2"); exit(1); }

/* Remove the semaphore */
if(semctl(semid,(int)NULL,IPC_RMID,NULL) == -1) perror("Parent >  semctl#1 >  ");

/* Detach from the shared mem */
if(shmdt(control) != 0) {
    perror("shmdt >  "); exit(1); }
    /* Remove the segment from the system */
    if(shmctl(shmid,IPC_RMID,NULL) != 0) {
        perror("shmctl >  "); exit(1); }

    printf("Parent >  Completed\n");

    exit(0);
CHILD:
    /* CHILD */

    /* Tell everyone I am waiting */
    printf("Child >  #%d waiting\n",identity);

    /* Count down the semaphore */
    if(semop(semid,sembuf_unlock,(size_t)1)!=0) {
        perror("Child >  semop #6 >  "); exit(1); }

    /* Wait for something to do */
    while(control->status != QUIT) {
        /* Is it me? */
       while (control->kid_number != identity && control->status != QUIT) usleep(100000);

        /* It is me, what do I have to do? */
        if(control->status == STATUS) {
            /* OK, just a status check */
            control->kid_number = -1;   /* Clear */
            control->status = OK;       /* Set OK */
            control->resp   = identity; /* Say who it is */
        }
    }

    /* Have found QUIT, bye bye */
    printf("Child >  #%d quitting\n",identity);

    /* Count down the semaphore */
    if(semop(semid,sembuf_unlock,(size_t)1)!=0) {
        perror("Child >  semop #6 >  "); exit(1); }

    exit(0);
}
void my_handler(signal, code, scp)
int             signal;
int             code;
struct sigcontext *scp;
{
    /* Have caught a signal -work on it */
    switch(signal) {
    case SIGUSR1:
        /* Increment the shared memory counter */
        control->resp++; break;
    case SIGUSR2:
        /* Say hello */
        printf("Child#%d says 'hello world'\n",identity); break;
    default:
        /* Where did that come from? */
        printf("Child#%d >  Unexpected signal -foo!\n",identity);
    }

    /* Count down the semaphore */
    if(semop(semid,sembuf_unlock,(size_t)1)!=0) {
        perror("Child >  semop #6 >  "); exit(1); }

    return;
```

```
}
```

The following example output shows the result in the correct output order:

```
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #3 waiting
Parent >  All children alive
Child#3 says 'hello world'
Child#2 says 'hello world'
Child#1 says 'hello world'
Child#0 says 'hello world'
Child#3 >  Unexpected signal -foo!
Child#2 >  Unexpected signal -foo!
Child#0 >  Unexpected signal -foo!
Child#1 >  Unexpected signal -foo!
parent >  all children responded!
Parent >  Children quit!
Child >  #3 quitting
Child >  #2 quitting
Child >  #1 quitting
Child >  #0 quitting
Parent >  Completed
```

7.9 Message queues

AIX 5L supports the System V style of message queue subroutines and these are listed in Table 65. For more information, please refer to *General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation.

Table 65. System V style message queue subroutines

Message subroutines	Description
msgget	Gets a message queue identifier.
msgsnd	Sends a message.
msgrcv	Reads a message from a queue.
msgxrcv	Receives an extended message.
msgctl	Provides message control operations.

The program message.c demonstrates communication between the parent and child processes using message queues:

```
/* message.c */
#include <stdio.h>  /* Needed for printf */
#include <sys/shm.h>    /* Needed for shmget */
#include <sys/types.h>  /* Needed for fork */
#include <unistd.h>     /* Needed for fork */
#include <sys/msg.h>    /* Needed for msgget... */
#include <string.h> /* Needed for strncmp */
```

```
/* maximum message size */
#define MAX_MSG_SIZE    64

#define NKIDS   5       /* How many children? */
#define NUM_OF_MSGS 5   /* How many predefined messages are there? */

struct mess {
    mtyp_t  mtype;
    char    mtext[MAX_MSG_SIZE];
} the_word;

char    *msgs[NUM_OF_MSGS]  = { "WE    must redo operation","TRIED operation again",
"REDs  Blues Greens", "GUN   condition hot",
"RANGE value undefined" };

int identity    = -1;           /* Define and initialize */
int ix  = -1;               /* Define and initialize */

main()
{

    pid_t   pid = -1;       /* Define and initialize */
    int     nkids = -1;         /* Define and initialize */
    int     child = 0;      /* Set child to false */
    int     parent = 0;         /* Set parent to false */
    key_t   mkey    = 0xf00;    /* Define and initialize */
    const   int shmsize = 8192;     /* Define and initialize */
    int     shmid   = -1;       /* Define and initialize */
    int itemp   = -1;
    int msgid   = -1;

    /* Initialise a message queue */
    itemp   = IPC_CREAT|S_IRUSR|S_IWUSR;
    if((msgid=msgget(IPC_PRIVATE,itemp))==-1) {
        perror("msgget #1"); exit(1); }

    printf("Parent >  Making children\n");

    /* Make NKIDS children */
    for (ix = 0; ix < NKIDS; ix++) {

        /* begat a child */
        if ((pid = fork()) == -1) {
            printf("fork #1"); exit(1); }

        /* Check to see if child or parent */
        if (pid == 0) {
            /* CHILD */
            child = 1;  /* Set child true */
            parent = 0;     /* and parent false -just to make sure */
            identity    = ix;   /* Set 'local' identity */
            goto CHILD;
        } else {
            /* PARENT */
            parent = 1;     /* I am the parent */
            child = 0;  /* not the child */
        }
    }

    /* PARENT SECTION */
    /* NKIDS children have been created */
    for (ix = 0; ix < NKIDS; ix++) {
        /* Read the message queue */
        if((itemp=msgrcv(msgid,&the_word,MAX_MSG_SIZE,ix+1,0)) == -1) {
            perror("Parent >  msgrcv #1 >  "); exit(1); }

        /* Check if the message is from the children */
        if(strncmp("Present",the_word.mtext,itemp) == 0) {
            printf("Child %d Present\n",ix);
        } else {
            printf("Parent >  unexpected message!\n");
        }
    }

    /* Write some messages out */
    /* The following splurge means use the minimum value */
    for (ix = 0; ix < (NKIDS>NUM_OF_MSGS?NUM_OF_MSGS:NKIDS); ix++) {

        /* Set the message type -Child id in this case */
        the_word.mtype = ix+ 1+ NKIDS;

        /* Set the message text */
        strcpy(the_word.mtext,msgs[ix]);

        /* Send the message */
        if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
            perror("Child >  msgsnd #1 >  "); exit(1); }

        /* Wait for child to get the message */
        usleep(10000);
    }

    /* Have spoken to everyone, tell all children to quit */
    printf("Parent >  Children quit!\n");

    /* Setup the quit message */
    strcpy(the_word.mtext,"Quit");

    for (ix = 0; ix < NKIDS; ix++) {

        /* Target each child in turn */
        the_word.mtype = ix+ 1+ NKIDS;

        /* Send the message */
        if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
            perror("Child >  msgsnd #2 >  "); exit(1); }
    }

    /* Wait for children to exit */
    usleep(20000);

    printf("Parent >  All children have quit, completing\n");

    /* Remove the message queue */
    if(msgctl(msgid,IPC_RMID,NULL) == -1) perror("Parent >  msgctl >  ");

    printf("Parent >  Completed\n");
    exit(0);
CHILD:
    /* CHILD */

    /* Tell everyone I am waiting */
    /* Setup the message type */
    the_word.mtype = identity+ 1;
    /* Setup the message text */
    strcpy(the_word.mtext,"Present");

    /* Send the message */
    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
        perror("Child >  msgsnd #3 >  "); exit(1); }

    printf("Child >  #%d waiting\n",identity);

    /* Wait for something to do */
    ix  = 1;
    while(ix) {
        /* Setup the message type first for this child */
        itemp   = identity+ 1+ NKIDS;

        /* Read the message queue */
        if((itemp=msgrcv(msgid,&the_word,MAX_MSG_SIZE,itemp,0)) == -1) {
            perror("Child >  msgrcv #2 >  "); exit(1); }

        /* Am I being told to quit? */
        if(strncmp("Quit",the_word.mtext,itemp) == 0) {
            ix  = 0;    /* Yes, set ix to a 'false' value */
        } else {
            printf("Child %d, received message: %s\n",identity,the_word.mtext);
        }
    }

    /* Have found QUIT, bye bye */
    printf("Child >  #%d quitting\n",identity);

    exit(0);
}
```

An example output is:

```
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #3 waiting
Child >  #4 waiting
Child 0 Present
Child 1 Present
Child 2 Present
Child 3 Present
Child 4 Present
Child 0, received message: RANGE value undefined
Child 1, received message: TRIED operation again
Child 2, received message: WE    must redo operation
Child 3, received message: GUN   condition hot
Child 4, received message: REDs  Blues Greens
Parent >  Children quit!
Child >  #0 quitting
Child >  #1 quitting
Child >  #2 quitting
Child >  #3 quitting
Child >  #4 quitting
Parent >  All children have quit, completing
Parent >  Completed
```

7.10 Timers and cyclic signals

When porting code, there is usually a time when it is good to understand how fast the machine is and where the CPU time is being used. Different UNIX-based implementations use several different methods of retrieving elapsed time from the system. In addition to timing your code, it may be necessary to run a section of code at a fixed frame rate (every n seconds or milliseconds and so forth). Table 66 shows the AES and System V style interfaces that facilitate timing and periodic interrupts that can be used within AIX 5L.

Table 66. Timer and cyclic interrupt subroutines

Time subroutines	Description
gettimer	Gets the current value for the specified system-wide timer.
getitimer, setitimer	Manipulates the expiration time of interval timers.

AIX 5L, Solaris 8, HP-UX 11, and Tru64 5.1 all support the functions listed with one exception: Solaris does not support the gettimer routine.

The timer.c example (shown here) demonstrates the use of gettimer and setitimer. The program uses setitimer to set up a cyclic interrupt to arrive at 20 millisecond intervals and trigger the handler. The routine foo_handler snaps the clock with gettimer, measures the time from the last interrupt, and prints it to stdout.

The code does not watch out for clock wrapping, so beware!

```
/* timer.c */
#include <stdio.h>  /* Needed for printf */
#include <sys/time.h>   /* Needed for gettimer */
#include <sys/types.h>  /* Needed for gettimer */
#include <unistd.h> /* Needed for setitimer */
#include <signal.h> /* Needed for sigemptyset... */

#define SECS_DELAY  0   /* Period of interrupt in seconds */
#define USECS_DELAY 20000   /* Period of interrupt in microseconds */
#define NINTS       10  /* Number of interrupts to grab before exit */

struct  timestruc_t the_time[2];    /* Define */

long    is  = -1;           /* Define and initialize */
long    ins = -1;           /* Define and initialize */
float   fthis_time  = -1.0;     /* Define and initialize */
float   flast_time  = -1.0;     /* Define and initialize */
volatile long   int_count   = 0;        /* Define and initialize */

main()
{
    extern void foo_handler();  /* Define interrupt handler */

    struct  itimerval   setup_clock, old_clock; /* Define */
    struct  timeval     time_setting;       /* Define */

    long    *old_handler    = (long *)-1;   /* Define and initialize */
    sigset_t    set, old_set;       /* Define */
    struct sigaction action, oaction;   /* define */

    sigemptyset(&set);      /* Get a clean set */
    sigaddset(&set,SIGALRM);    /* Add SIGALARM */

    /* Setup the handler */
    action.sa_handler = &foo_handler;

    /* Setup signal handler */
    if (sigaction(SIGALRM, &action, &oaction) != 0) {
        perror("sigaction >  "); exit(1); }

    /* Clear the time counters */
    fthis_time  = 0.0;
    flast_time  = 0.0;

    /* Get the time */
    if(gettimer(TIMEOFDAY,&the_time[0]) != 0) {
        perror("gettimer#1 >  "); exit(1); }

    /* Setup the interrupt characteristics */
    time_setting.tv_sec = SECS_DELAY;
    time_setting.tv_usec    = USECS_DELAY;

    /* Setup the control structure */
    setup_clock.it_interval = time_setting;
    setup_clock.it_value    = time_setting;

    /* OK, ready to go, start the interrupt */
    if(setitimer(ITIMER_REAL,&setup_clock,&old_clock) != 0) {
        perror("setitimer >  "); exit(1); }

    /* Do the processing loop until we have hit NINTS */
    while(int_count<NINTS) { /* Maybe check a variable set by the interrupt handler */
                             /*
                             * Do lots of magic processing here
        */
        sleep(100); /* Sleep here instead of burning CPU time */
    }
}
void foo_handler(signal, code, scp)
int             signal;
int             code;
struct sigcontext *scp;
{
    /* Snap the clock */
    if(gettimer(TIMEOFDAY,&the_time[1]) != 0) {
        perror("gettimer#2 >  "); exit(1); }

    /* Increment the interrupt count */
    int_count++;

    is  = the_time[1].tv_sec- the_time[0].tv_sec;
    ins = the_time[1].tv_nsec- the_time[0].tv_nsec;

    /* Get the time value */
    fthis_time  = is+ (ins/1E9);

    printf("foo_handler > Interval is %.3f seconds\n", fthis_time-flast_time);
    /* Remember what the time is */
    flast_time  = fthis_time;

    return;
}
```

An example output of timer.c is:

```
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
foo_handler > Interval is 0.020 seconds
```

Chapter 8. The compilers

This chapter covers the IBM C and C++ compilers available for the AIX 5L operating system on the Power platform. The compilers are perhaps the most important part of your development environment. In this chapter, we describe the features of the IBM C for AIX Version 5 C compiler and the IBM VisualAge C++ Professional for AIX Version 5 compiler.

8.1 The C compiler

This section covers the IBM C for AIX Version 5 compiler. Version 5.0.2 of this compiler is required to support AIX 5L.

8.1.1 C for AIX 5L compiler limits

Table 67 lists some compiler limits that may be of interest to large complex programs. In general, for the sake of the sanity of the developer, most application programs use identifiers with significantly less than 4095 characters.

Table 67. Compiler limits

Language Feature	Limit
Nesting levels for included files	255
Significant initial characters in identifiers	No limit (but the linker has a limit of 4095 characters for external names)

Other system limits are set in the /usr/include/sys/limits.h file. The limits.h header file is described in the *AIX 5L Version 5.1 Files Reference*, which can be found in the AIX 5L online documentation.

8.1.2 Environment variables affecting the compilers

As with most applications in a UNIX environment, there are environment variables that affect the compiler. The compiler will respond to the environment variables that are covered in the sections below.

8.1.2.1 PATH environment variable

The most basic environment variable that affects the compiler is the same for all other applications that are invoked from the command line, the PATH environment variable. Section 4.9.3.1, “Finding the compiler drivers” on page 100 describes how you can correctly set up this variable.

8.1.2.2 OBJECT_MODE environment variable

The OBJECT_MODE variable is used to indicate the preferred object mode that should be used by various development tools (such as ar, ld, dump, and so on). The variable changes the default compilation mode behavior, unless it is overridden by the compiler configuration file or command line options. The actual relationship between the value of OBJECT_MODE and the 32/64 bit mode that the compiler will compile code to is depicted in Table 68, which assumes that no command line option or configuration file overrides the setting.

Table 68. OBJECT_MODE settings and the compiler behavior

OBJECT_MODE	Compilation-mode behavior
Not set	32-bit compiler mode
32	32-bit compiler mode
62	64-bit compiler mode
32_64	Fatal error and stop with the following message: `1501-054 OBJECT_MODE=32_64 is not a valid setting for the compiler`
Any other	Fatal error and stop with the following message: `1501-255 OBJECT_MODE setting is not recognized and is not a valid setting for the compiler`

8.1.2.3 Return codes, warning, and error messages

Generally, you want your compiler to be quiet when compiling, but when your program contain errors or potential errors, you want a message that states what is wrong or potentially wrong in the program.

The compiler will produce different messages depending on what it encounters in the program it is compiling. It will produce five different types of messages:

- Informational
- Warning
- Error
- Severe error
- Unrecoverable error

The messages will have a format similar to the one described in Figure 46 on page 221.

```
15cc-nnn (severity) text.

where:
     cc         Is a two-digit code that tells which component issued the message:
                00 Code-generation or optimization message
                01 Compiler services message
                05 Front-end text message
                06 Front-end error message
                40 - message specific to C for AIX compiler
                41 - message specific to C for AIX compiler
                46 - message specific to C for AIX compiler backend
                86 - message specific to interprocedural analysis (IPA).
     nnn
                Is the message number
     severity
                Is a letter representing the severity level of the message
     text
                Is the message text describing the error
```

Figure 46. Compiler diagnostics message format

The letters in the severity section of Figure 46 correspond to the different message types. These are specified in Table 69, where you also can see how the compiler will react after putting out a message.

Table 69. Diagnostic messages their severity and the compiler response

Letter	Severity	Compiler Response
I	Informational	Compilation continues. The message reports conditions found during compilation.
W	Warning	Compilation continues. The message reports valid, but possibly unintended, conditions.
E	Error	Compilation continues and object code is generated. Error conditions exist that the compiler can correct, but the program might not run correctly.
S	Severe error	Compilation continues, but object code is not generated. Error conditions exist that the compiler cannot correct.
U	Unrecoverable error	The compiler halts. An internal compiler error has been found. This message should be reported to your IBM service representative.

In Figure 47 on page 222, you can see an example of the messages the compiler will give you when compiling a program called a.c with a missing semi-colon (;) in line 8. In this case, the return code from the compiler is 1.

```
$ cc -o fltest a.c
"a.c", line 8.1: 1506-277 (S) Syntax error: possible missing ';' or ','?
"a.c", line 10.1: 1506-045 (S) Undeclared identifier x.
$ echo $?
1
$
```

Figure 47. A severe error message

If you specify the compiler option -qsrcmsg and the error is applicable to a particular line of code, the reconstructed source line or partial source line is included with the error message in the stderr file. A reconstructed source line is a preprocessed source line that has all the macros expanded. An example of this is shown in Figure 48.

```
$ cc -o fltest fltest.c -qsrcmsg
        8 | double x,y,z,v
            a.............
a - 1506-277 (S) Syntax error: possible missing ';' or ','?
       10 | x=1.0;
            a.....
a - 1506-045 (S) Undeclared identifier x.
$ echo $?
1
$
```

Figure 48. Severe error message displayed with the -qsrcmsg flag

The reconstructed source line represents the line as it appears after macro expansion. At times, the line may be only partially reconstructed. The characters "...." at the start or end of the displayed line indicate that some of the source line has not been displayed.

As you can see in Figure 46 on page 221 and Table 69 on page 221, the return code from the compiler was 1. The meaning of the return code from the compiler can be seen in Table 70.

Table 70. Error types and return codes

Return code	Error type
1	Any error with a severity level higher than the setting of the halt compiler option has been detected.
40	An option error or an unrecoverable error has been detected.

Return code	Error type
41	A configuration file error has been detected.
250	An out-of-memory error has been detected. The `xlc` command cannot allocate any more memory for its use.
251	A signal-received error has been detected, that is, an unrecoverable error or interrupt signal has occurred.
252	A file-not-found error has been detected.
253	An input/output error has been detected: files cannot be read or written to.
254	A fork error has been detected. A new process cannot be created.
255	An error has been detected while the process was running.

One error that you might encounter if you are compiling *very* big programs and using heavy optimization is the following:

```
1501-229 Compilation ended due to lack of space.
1501-224 fatal error in ../exe/xlCcode: signal 9 received.
```

This is caused by the AIX 5L operating system running low on paging space. If lack of paging space causes other compiler programs to fail, the following message may be displayed:

```
Killed.
```

To minimize paging space problems, do any of the following and recompile your program:

- Reduce the number of processes competing for system paging space.
- Increase the system paging space.
- Compile your program without optimization.
- Reduce the size of your program by splitting it into two or more source files.

To check the current paging-space settings enter the command `lsps -a` or use the AIX System Management Interface Tool (SMIT) command `smit pgsp`.

The paging-space overview in *AIX 5L Version 5.1 System User's Guide: Operating System and Devices* section, which can be found in the AIX 5L online documentation, describes paging space and how to allocate it.

8.1.3 Types of input files

You can input the following types of files to the C for AIX compilers.

8.1.3.1 C source files

These are files containing a C source module. The source file must have a .c (lowercase c) suffix, for example, foo.c. A source file could look like the one that we will use as an example, shown in Figure 49.

```
#include "../inc/foo.h"

extern void bar(void);

void foo(void)
{
  printf(FOO);
}

main() {
  foo();
  bar();
  printf("\n");
}
```

Figure 49. A very simple source file (foo.c)

The compiler will also accept source files with the .i suffix. This extension indicates that the file is a preprocessed source file.

The compiler processes the source files in the order in which they appear on the command line. If the compiler cannot find a specified source file, it produces an error message and the compiler proceeds to the next specified file. However, the link editor will not be run and temporary object files will be removed.

Your program can consist of several source files. All of these source files can be compiled at once using only one invocation of the compiler. Although more than one source file can be compiled using a single invocation of the compiler, you can specify only one set of compiler options on the command line per invocation. Each distinct set of command line compiler options that you want to specify requires a separate invocation.

By default, the compiler preprocesses and compiles all the specified source files. Although you will usually want to use this default, you can preprocess the source file without compiling by specifying either the -E or the -P option. If you specify the -P option, a preprocessed source file (foo.i) is created, and processing ends.

The -E option preprocesses the source file, writes the result to standard output, and halts processing without generating an output file.

8.1.3.2 Preprocessed source files
Preprocessed source files have a .i suffix, for example, foobar.i. The compiler driver command sends the preprocessed source file (foobar.i) to the compiler, where it is preprocessed again in the same way as a .c file. Preprocessed files are useful for checking macros and preprocessor directives.

8.1.3.3 Object files
Object files must have a .o suffix, for example, foo.o. Object files, library files, and non-stripped executable files serve as input to the linkage editor.

After compilation, the linkage editor links all of the specified object files to create an executable file.

8.1.3.4 Assembler files
Assembler files must have a .s suffix, for example, bar.s. Assembler files are assembled to create an object file. The assembler version of the bar.c would be similar to the example shown in Figure 50 on page 226.

```
#lots of set's deleted

        .rename H.11.NO_SYMBOL{PR},""
        .rename E.17.__STATIC{RW},"_$STATIC"
        .rename H.19.__STATIC{TC},"_$STATIC"
        .rename H.23.bar{TC},"bar"

        .lglobl H.11.NO_SYMBOL{PR}
        .globl  .bar
        .lglobl E.17.__STATIC{RW}
        .globl  bar{DS}
        .extern .printf{PR}

# .text section
        .file   "bar.c"

        .csect  H.11.NO_SYMBOL{PR}
.bar:                                   # 0x00000000 (H.11.NO_SYMBOL)
        stu     SP,-64(SP)
        mfspr   r0,LR
        l       r3,T.19.__STATIC(RTOC)
        st      r0,72(SP)
        bl      .printf{PR}
        oril    r0,r0,0x0000
        l       r12,72(SP)
        cal     SP,64(SP)
        mtspr   LR,r12
        bcr     BO_ALWAYS,CR0_LT
        .long   0x00000000
# traceback table
        .byte   0x00                    # VERSION=0
        .byte   0x00                    # LANG=TB_C
        .byte   0x20                    # IS_GL=0,IS_EPROL=0,HAS_TBOFF=1
                                        # INT_PROC=0,HAS_CTL=0,TOCLESS=0
                                        # FP_PRESENT=0,LOG_ABORT=0
        .byte   0x01                    # INT_HNDL=0,NAME_PRESENT=0
                                        # USES_ALLOCA=0,CL_DIS_INV=WALK_ONCOND
                                        # SAVES_CR=0,SAVES_LR=1
        .byte   0x80                    # STORES_BC=1,FPR_SAVED=0
        .byte   0x00                    # GPR_SAVED=0
        .byte   0x00                    # FIXEDPARMS=0
        .byte   0x00                    # FLOATPARMS=0,PARMSONSTK=0
        .long   0x00000028              # TB_OFFSET
# End of traceback table
# End   csect   H.11.NO_SYMBOL{PR}
# .data section

        .toc                            # 0x00000038
T.23.bar:
        .tc     H.23.bar{TC},bar{DS}
T.19.__STATIC:
        .tc     H.19.__STATIC{TC},E.17.__STATIC{RW}


        .csect  bar{DS}
        .long   .bar                    # "\0\0\0\0"
        .long   TOC{TC0}                # "\0\0\0008"
        .long   0x00000000              # "\0\0\0\0"
# End   csect   bar{DS}

        .csect  E.17.__STATIC{RW}
        .long   0x62617200              # "bar\0"
# End   csect   E.17.__STATIC{RW}
# .bss section
```

Figure 50. PowerPC assembler version of the bar.c

8.1.3.5 Non-stripped executable files

Extended Common Object File Format (XCOFF) files that have not been stripped with the AIX `strip` command can be used as input to the compiler.

See the `strip` command in the *AIX 5L Version 5.1 Commands Reference* and the description of a.out file format in the *AIX 5L Version 5.1 Files Reference*, both of which can be found in the online documentation, for more information.

8.1.4 Output files

The C compiler will generate different types of output files, all depending on what you ask it to do. These output files fall into different categories that are covered in this section.

8.1.4.1 Executable file

By default, executable files are named a.out. To name the executable file something else, use the -o <filename> option with the `invocation` command. This option creates an executable file, with the name you specify as <filename>. The name you specify can be a relative or absolute path name for the executable file. The format of the a.out file is described in the *AIX 5L Version 5.1 Files Reference*, which can be found in the online documentation.

8.1.4.2 Object files

Object files must have an .o suffix, for example, foo.o, unless the -o <filename> option is specified. If you specify the -c option, an output object file, <filename>.o, is produced for each input source file foobar.c. The linkage editor is not invoked, and the object files are placed in your current directory. All processing stops at the completion of the compilation. You can link-edit the object files later into a single executable file using the `compiler` command, or, alternatively, invoke the linker directly.

8.1.4.3 Assembler files

Assembler files must have a .s suffix, for example, foo.s. They are created by specifying the -S option. Assembler files can then be assembled to create an object file. An example of an assembler file can be seen in Figure 50 on page 226.

8.1.4.4 Preprocessed source files

Preprocessed source files have a .i suffix, for example, foo.i. To make a preprocessed source file, specify the -P option. The source files are preprocessed but not compiled. A preprocessed source file, foo.i, is generated for each source file, foo.c. Figure 51 on page 228 shows an example of a preprocessed source file.

```
$ cat foo.c
#include "../inc/foo.h"

/*
 * The bar function is located in the bar.c file.
 */

extern void bar(void);

void foo(void)
{
  printf(FOO);
}

main() {

  foo(); /* call foo */
  bar(); /* call bar */
  printf("\n");
}
$ cc -P foo.c
$ cat foo.i



extern void bar(void);

void foo(void)
{
  printf("foo");
}

main() {

  foo();
  bar();
  printf("\n");
}
```

Figure 51. A preprocessed .i file

8.1.4.5 Listing files

Listing files have an .lst suffix, for example, foobar.lst. Specifying any one of the listing-related options to the `invocation` command produces a compiler listing (unless you have specified the -qnoprint option). The file containing this listing is placed in your current directory and has the same file name (with a .lst extension) as the source file from which it was produced.

8.1.4.6 Target file

Output files associated with the -M option have a .u suffix, for example, conversion.u. The file contains targets suitable for inclusion in a description file for the `make` command. A .u file is created for every input file with a .c or .i suffix. .u files are not created for any other files (unless you use the -+ option so other file suffixes are treated as .c files). An example of the creation of a .u file can be seen in Figure 52. As you can see, the foo.o file depends on ../inc/foo.h and foo.c.

```
$ cat foo.c
#include "../inc/foo.h"
extern void bar(void);

void foo(void)
{
  printf(FOO);
}

main() {
  foo();
  bar();
  printf("\n");
}
$ cc -c -M foo.c
$ cat foo.u
foo.o: foo.c
foo.o: ../inc/foo.h
$
```

Figure 52. Using the -M flag to generate a .u target file

8.1.4.7 Static libraries

The compiler does not produce static libraries, but we mention them here anyway because all that really happens to the object files after the compiler has finished is that the ar archiver makes an archive of them. For further information on how static libraries work under AIX 5L, refer to Section 9.1.1, “Static library” on page 258.

8.1.4.8 Shared libraries

Shared libraries are described in Section 9.1.2, “Shared library” on page 258.

8.1.5 Type conversions

Type conversions are implementation dependent. Because of this, you might have to rewrite some code. Have a look trough the tables in this section; they summarize type conversions of arithmetic types. Arithmetic types include signed and unsigned integral types (char, int, short and long) in addition to float, double, and long double types. In Table 71 you can see how to convert into signed types.

Table 71. Type conversions to signed integer types

To:	**signed char**	**signed short**	**signed int**	**signed long**	**signed long long**
From:					
signed char	None.	Sign extend.	Sign extend.	Sign extend.	Sign extend.
signed short	Preserve low-order bytes.	None.	Sign extend.	Sign extend.	Sign extend.
signed int	Preserve low-order bytes.	Preserve low-order bytes.	None.	Preserve bit pattern.	Sign extend.
signed long	Preserve low-order bytes.	Preserve low-order bytes.	Preserve low-order bytes.	None.	Sign extend.
signed long long	Preserve low-order bytes.	Preserve low-order bytes.	Preserve low-order bytes.	Preserve low-order bytes.	None.
unsigned char	Preserve bit pattern; high-order bit becomes sign bit.	Zero extend.	Zero extend.	Zero extend.	Zero extend.
unsigned short	Preserve low-order bytes.	Preserve bit pattern; high-order bit becomes sign bit.	Zero extend.	Zero extend.	Zero extend.
unsigned int	Preserve low-order bytes.	Preserve low-order bytes.	Preserve bit pattern; high-order bit becomes sign bit.	Preserve bit pattern; high-order bit becomes sign bit.	Zero extend.
unsigned long	Preserve low-order bytes.	Preserve low-order bytes.	Preserve bit pattern; high-order bit becomes sign bit.	Preserve bit pattern; high-order bit becomes sign bit.	

To:	signed char	signed short	signed int	signed long	signed long long
From:					
unsigned long long	Preserve low-order bytes.	Preserve low-order bytes.	Preserve low-order bytes.	Preserve low-order bytes.	Preserve bit pattern; high-order bit becomes sign bit.
float	Convert to int, and convert int to signed char.	Convert to int, and convert int to signed short.	Truncate at decimal. If result is too large for int, result is undefined.	Truncate at decimal. If result is too large for long, result is undefined.	Truncate at decimal. If result is too large for long long, result is undefined.
double	Convert to int, and convert int to signed char.	Convert to int, and convert int to signed short.	Truncate at decimal. If result is too large for int, result is undefined.	Truncate at decimal. If result is too large for long, result is undefined.	Truncate at decimal. If result is too large for long long, result is undefined.
long double	Convert to int, and convert int to signed char.	Convert to int, and convert int to signed short.	Truncate at decimal. If result is too large for int, result is undefined.	Truncate at decimal. If result is too large for long, result is undefined.	Truncate at decimal. If result is too large for long long, result is undefined.

Table 72 shows how to convert into unsigned types.

Table 72. Type conversions to unsigned Integer types

To:	unsigned char	unsigned short	unsigned int	unsigned long	unsigned long long
From:					
signed char		Sign extend to short, and convert to unsigned short	Sign extend to int, and convert int to unsigned int	Sign extend to long, and convert long to unsigned long	Sign extend to long long, and convert long long to unsigned long long

To:	unsigned char	unsigned short	unsigned int	unsigned long	unsigned long long
From:					
short	Preserve low-order byte	Preserve bit pattern; sign function of sign bit lost	Preserve bit pattern; sign function of sign bit lost Sign extend to int, and convert int to unsigned int	Sign extend to long, and convert long to unsigned long	Sign extend to long long, and convert long long to unsigned long long
int	Preserve low-order byte	Preserve low-order byte	Preserve bit pattern; sign function of sign bit lost	Preserve bit pattern; sign function of sign bit lost	Sign extend to long long, and convert long long to unsigned long long
long	Preserve low-order byte	Preserve low-order byte	Preserve bit pattern; sign function of sign bit lost	Preserve bit pattern; sign function of sign bit lost	Sign extend to long long, and convert long long to unsigned long long
long long	Preserve low-order byte	Preserve low-order byte	Preserve low-order byte	Preserve low-order byte	Preserve bit pattern; sign function of sign bit lost
unsigned char	None	Zero extend	Zero extend	Zero extend	Zero extend
unsigned short	Preserve low-order bytes	None	Zero extend	Zero extend	Zero extend
unsigned int	Preserve low-order bytes	Preserve low-order bytes	None	Preserve bit pattern	Zero extend
unsigned long	Preserve low-order bytes	Preserve low-order bytes	Preserve bit pattern	None	Zero extend

To:	unsigned char	unsigned short	unsigned int	unsigned long	unsigned long long
From:					
unsigned long long	Preserve low-order bytes	Preserve low-order bytes	Preserve low-order bytes	Preserve low-order bytes	None
float	Convert to int, and convert int to unsigned char	Convert to unsigned int, and convert unsigned int to unsigned short	Truncate; if result is negative or too large, the result is undefined	Truncate; if result is negative or too large, the result is undefined	Truncate; if result is negative or too large, the result is undefined
double	Convert to int, and convert int to unsigned char	Convert to unsigned int, and convert unsigned int to unsigned short	Truncate; if result is negative or too large, the result is undefined	Truncate; if result is negative or too large, the result is undefined	Truncate; if result is negative or too large, the result is undefined
long double	Convert to int, and convert int to unsigned char	Convert to unsigned int, and convert unsigned int to unsigned short	Truncate; if result is negative or too large, the result is undefined	Truncate; if result is negative or too large, the result is undefined	Truncate; if result is negative or too large, the result is undefined

Table 73 shows the conversion to floating-point types.

Table 73. Type conversions to floating-point types

To:	float	double	long double
From:			
signed char	Sign extend to int, and convert int to float	Sign extend to int, and convert int to double	Sign extend to int, and convert int to long double
signed short	Sign extend to int, and convert int to float	Sign extend to int, and convert int to double	Sign extend to int, and convert int to long double

To:	float	double	long double
From:			
signed int	Represent as float; if the int cannot be represented exactly, some loss of precision may occur	Represent as double; if the int cannot be represented exactly, some loss of precision may occur	Represent as long double; if the int cannot be represented exactly, some loss of precision may occur
signed long	Represent as float; if the long cannot be represented exactly, some loss of precision may occur	Represent as double; if the long cannot be represented exactly, some loss of precision may occur	Represent as long double; if the long cannot be represented exactly, some loss of precision may occur
signed long long	Represent as float; if the long long cannot be represented exactly, some loss of precision may occur	Represent as double; if the long long cannot be represented exactly, some loss of precision may occur	Represent as long double; if the long long cannot be represented exactly, some loss of precision may occur
unsigned char	Sign extend to int, and convert int to float	Sign extend to int, and convert int to double	Sign extend to int, and convert int to long double
unsigned short	Sign extend to int, and convert int to float	Sign extend to int, and convert int to double	Sign extend to int, and convert int to long double
unsigned int	Represent as float; if the int cannot be represented exactly, some loss of precision may occur	Represent as double; if the int cannot be represented exactly, some loss of precision may occur	Represent as long double; if the int cannot be represented exactly, some loss of precision may occur
unsigned long	Represent as float; if the long cannot be represented exactly, some loss of precision may occur	Represent as double; if the long cannot be represented exactly, some loss of precision may occur	Represent as long double; if the long cannot be represented exactly, some loss of precision may occur

To:	float	double	long double
From:			
unsigned long long	Represent as float; if the long long cannot be represented exactly, some loss of precision may occur	Represent as double; if the long long cannot be represented exactly, some loss of precision may occur	Represent as long double; if the long long cannot be represented exactly, some loss of precision may occur
float	None	Convert to double	Convert to long double
double	Represent as float; if result is too large, result is undefined	None	Convert to long double
long double	Convert to float	Represent as double; if result is too large to be represented as double, result is undefined	None

8.1.5.1 Converting pascal string literals
The -qmacpstr option converts Pascal string literals of the form "\pABC" into null-terminated strings, where the first byte contains the length of the string.

8.1.5.2 Integral promotion
The default compiler action is for integral promotions to convert a char, short int, int bit field or their signed or unsigned types, or an enumeration type, to an int. Otherwise, the type is converted to an unsigned int.

The -qupconv option promotes any unsigned type smaller than an int to an unsigned int instead of to an int.

8.1.5.3 Registers
Objects in registers declared with the storage class specifier register are treated as int objects.

8.1.6 C compiler files and directories
This section should give you a feel of where the different parts of the C compiler are installed. In Table 74 on page 236, we have the directory structure of the C compiler. The C compiler is installed in the /usr/vac directory structure, and has configuration files in /etc/, the include files in

/usr/include, and the library files in /usr/lib/. You can see the full tree in Table 74.

Table 74. Directory structure of the C compiler

Description	**Directory**
Configuration files for the C compiler	/etc/
Directory that holds the C compiler binaries.	/usr/vac/bin/
Root directory for the C compiler	/usr/vac/
Header files	/usr/include/ /usr/vac/include/
Library files (This depends on what you actually have installed on the machine.)	/usr/lib/ /usr/vac/lib/ /usr/vac/lib/<os version>
Message files	/usr/lib/nls/msg/$LANG/
Directory that holds the C compiler tools such as the preprocessor, assembler, and disassembler	/usr/vac/exe/
C compiler documentation	/usr/vac/html/

The programs in /usr/vac/exe/ are not intended to be used from the command line, but if needed, it can be done. Figure 53 on page 237 shows how you can use /usr/vac/exe/dis to disassemble an executable, if you so desire. But generally, you should use the standard interface, rather than calling for example the dissembler directly.

```
$ ls -l
total 13
-rwxr-xr-x   1 jasper   usr           6403 Mar 22 09:49 hwinfo
$ /usr/vac/exe/dis hwinfo
$ ls -l
total 73
-rwxr-xr-x   1 jasper   usr           6403 Mar 22 09:49 hwinfo
-rw-r--r--   1 jasper   usr          30550 Mar 22 09:51 hwinfo.s
$ head -3 hwinfo.s
.set r0,0; .set SP,1; .set RTOC,2; .set r3,3; .set r4,4
.set r5,5; .set r6,6; .set r7,7; .set r8,8; .set r9,9
.set r10,10; .set r11,11; .set r12,12; .set r13,13; .set r14,14
$
```

Figure 53. Disassembling a program

The default links in /usr/bin (xlc, cc, c89, etc) that point to /usr/vac/bin/xlc of C for AIX are optional. They are created at the discretion of the product installer using `replaceCSET`.

Table 75. Files used by the C compiler

Description	**File**
C for AIX README file, which contains important information not included in other documentation. Read this file before you use the compiler for the first time.	/usr/vac/xlC/README.C
C front end.	/usr/vac/exe/xlcentry
Help file.	/usr/vac/exe/default_msg/vac.help
C preprocessor.	/usr/vac/exe/xlCcpp
Disassembler	/usr/vac/exe/dis
Interprocedural analysis tool.	/usr/vac/exe/ipa
Code generator.	/usr/vac/exe/xlCcode /usr/vac/exe/bolt
C driver programs.	/usr/vac/bin/xlc /usr/vac/bin/xlc128 /usr/vac/bin/xlc_r /usr/vac/bin/cc /usr/vac/bin/cc128 /usr/vac/bin/cc_r

Description	File
Precompiled header support.	/usr/vac/lib/compmalloc.o
Memory debug support.	/usr/vac/lib/libhm.a /usr/vac/lib/libhm_r.a /usr/vac/lib/libhmd.a /usr/vac/lib/libhmd_r.a /usr/vac/lib/libhu.a /usr/vac/lib/libhu_r.a /usr/vac/include/stdlib.h /usr/vac/include/string.h /usr/vac/include/umalloc.h
Profile-directed feedback library.	/usr/vac/lib/libpdf.a
Profiling library.	/usr/vac/lib/profiled
Configuration files.	/etc/vacpp.cfg.<oslevel> /etc/vac.cfg.<oslevel>
Link to actual used configuration file.	/etc/vac.cfg
Links to /usr/vac/bin.	/usr/bin/xlc /usr/bin/xlc128 /usr/bin/xlc_r /usr/bin/c89 /usr/bin/cc /usr/bin/cc128 /usr/bin/cc_r
Other executables used by the compiler.	/bin/as /bin/ld /bin/strip

8.1.7 Command line arguments

The thing many programmers find the most confusing, or even irritating, when translating compiler options from one platform to another is, “What is this option now called and how come the syntax is made in this way, and not in the way I am used to?”

Normally you know what options you want to use, or rather what you want to do. But what you do not know is the new syntax and what the default options are. So we tried to list and compare all the options for the C compilers for AIX, HP-UX, Solaris and Tru64 in Appendix C, “C compiler options” on page 527. If you are using the GNU compilers, we suggest that you continue to use them as part of the porting process so there is no need to migrate to the IBM compilers.

To make it easier for you to find the options you need to translate, we have split the options into groups that correspond with the subsection headings used in Section 8.1.7, “Command line arguments” on page 238. So when you read Section 8.1.7.4, “Optimization+ and performance compiler options” on page 240, you might want to browse the table in Appendix C.3, “Optimization and performance compiler options” on page 529 to compare the options for all the operating systems.

We have chosen not to make a translation table that lets you mechanically translate option -oldopt to -newopt, because such tables normally just make a port longer. So look at this as an excuse to have a closer look at what the C compiler on AIX 5L offers; you will almost certainly find a new option or two you can use.

It might be a bit difficult to find the option you are looking for because on some platforms, to make your program use parallelization is an optimization option and on others it is a parallelization option.

For example, the parallelization options for the C compiler on HP-UX are normally found under the optimization flags in the manual. But For AIX 5L, Solaris, and Tru64, the manuals put them under parallelization. So we have listed those options under parallelization and not under optimization. We feel this is the best possible compromise.

8.1.7.1 Default compiler options

The default compiler options are defined in the configuration files for the compiler. The location of these files are described in Section 8.1.6, “C compiler files and directories” on page 235.

8.1.7.2 Licensing compiler options

As you can see in Appendix C.1, “Licensing compiler options” on page 527, only the C compiler under Solaris supports licensing options, so if you are porting from Solaris to AIX 5L you can forget all about these options.

8.1.7.3 Standard compliance compiler options

One thing that really can help you when porting applications between platforms is the ability to choose the level of standards compliance.

The C for AIX C compiler conforms to the following industry standards for compiling C language source code:

- The Federal Information Processing Standard (FIPS) PUB 160 C language.

- The American National Standard for Information Systems (ANSI) and International Standards Organization (ISO) standard ANSI/ISO-IEC 9899-1990[1992] for the C programming language.
- The International Standards Organization (ISO) standard ISO/IEC 9899:1990(E) for the C programming language.

A table that contains the AIX 5L Standard compliance compiler options and the corresponding ones for Solaris, HP-UX and Tru64 can be found in Appendix C.2, “Standards compliance compiler options” on page 527.

8.1.7.4 Optimization+ and performance compiler options

All the compilers support different methods of optimization, and some of the compilers have quite a lot of options. We have tried to split the options up into smaller portions.

Optimization techniques used by the C for AIX compiler include:

Aliasing	Aliasing is a compiler term for a storage location having multiple variables that reference it. When potential aliases occur, they inhibit the assumptions a compiler can make when optimizing a program.
Value numbering	Involves constant propagation, expression elimination, and folding of several instructions into a single instruction.
Branch optimizations	Rearranges the program code to minimize branching logic and to physically combine separate blocks of code.

Elimination	In common expressions, the same value is recalculated in a subsequent expression. The duplicate expression can be eliminated by using the previous value. This step is done even for intermediate expressions within expressions. For example, if your program contains the following statements:
	`a = c + d;`
	`...`
	`f = c + d + e;`
	the common expression c + d is saved from its first evaluation and is used in the subsequent statement to determine the value of f.

Code motion	If variables used in a computation within a loop are not altered within the loop, the calculation can be performed outside of the loop and the results used within the loop.
Invariant IF code floating	Removes invariant branching code from loops to make more opportunity for other optimizations.
Reassociating	Rearranges the sequence of calculations in an array-subscript expression, producing more candidates for common-expression elimination.
Strength reduction	Replaces less efficient instructions with more efficient ones. For example, in array subscripting, an add instruction replaces a multiply instruction.
Constant propagation	Constants used in an expression are combined, and new ones are generated. Some implicit conversions between integer and floating-point types are done.
Store motion	Moves store instructions out of loops.
Dead store elimination	Eliminates stores when the value stored is never referred to again. For example, if two stores to the same location have no intervening load, the first store is unnecessary and is removed.
Dead code elimination	Eliminates code that cannot be reached or code whose results are not subsequently used.
Inlining	Replaces function calls with actual program code.
Instruction scheduling	Reorders instructions to minimize execution time.
Interprocedural analysis	Uncovers relationships across function calls, and eliminates loads, stores, and computations that cannot be eliminated with more straightforward optimizations.
Global register allocation	Allocates variables and expressions to available hardware registers using a graph coloring algorithm.

A table that contains the AIX 5L compiler optimization options and the corresponding ones for Solaris, HP-UX and Tru64 can be found in Appendix C.3, “Optimization and performance compiler options” on page 529.

8.1.7.5 Data alignment compiler options

A table that contains the AIX 5L compiler data alignment compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.4, “Data alignment compiler options” on page 544.

8.1.7.6 Floating point and numeric compiler options

A table that contains the AIX 5L compiler floating point and numeric compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.5, “Floating point and numeric compiler options” on page 546.

8.1.7.7 Parallelization compiler options

Parallelization options deal with the use of the OpenMP API and the IBM parallelization directives. These are described in Chapter 5, “Program Parallelization” in the *C for AIX User’s Guide*, which is part of the compiler’s online documentation.

Furthermore, they also deal with threads, which is described in Chapter 10, “POSIX threads” on page 307.

A table that contains the AIX 5L compiler target platform compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.6, “Parallelization compiler options” on page 552.

8.1.7.8 Source code compiler options

A table that contains the AIX 5L compiler source code compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.7, “Source Code compiler options” on page 553.

8.1.7.9 Compiled code compiler options

A table that contains the AIX 5L compiler compiled code compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.8, “Compiled code compiler options” on page 559.

8.1.7.10 Compilation mode compiler options

A table that contains the AIX 5L compiler compilation mode compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.9, “Compilation mode compiler options” on page 562.

8.1.7.11 Diagnostics compiler options

A table that contains the AIX 5L compiler diagnostics compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.10, “Diagnostics compiler options” on page 564.

8.1.7.12 Debugging compiler options

A table that contains the AIX 5L compiler debugging compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.11, “Debugging compiler options” on page 569.

8.1.7.13 Libraries compiler options

A detailed description on how libraries work under AIX 5L can be found in Chapter 9, “AIX shared objects and libraries” on page 257. In addition, a table that contains the AIX 5L compiler libraries compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.12, “Linking and libraries compiler options” on page 572.

8.1.7.14 Target platform compiler options

Due to the fact that the hardware platform evolves over time, you are forced to specify compiler options that take this into account. The target platform compiler options have several dimensions. These dimensions are:

- 32-bit or 64-bit
- Processor architecture
- Physical machine issues

A table that contains the AIX 5L compiler target platform compiler options and the corresponding ones for Solaris, HP-UX, and Tru64 can be found in Appendix C.13, “Target platform compiler options” on page 578.

8.1.8 Predefined preprocessor macros

The AIX 5L C preprocessor has two kinds of predefined preprocessor macros: the ones that are defined by the ANSI standard for the C programming language (shown in Table 76 on page 244), and the ones that are defined by the AIX 5L C compiler (for the purpose of detecting compile time environment characteristics). These can be seen in Table 77 on page 245.

Table 76. ANSI standard predefined preprocessor macros

Predefined macro	Description
__LINE__	An integer describing the current source line number. The value of __LINE__ changes during compilation as the compiler processes subsequent lines of your source program. It can be set with the `#line` directive.
__FILE__	A character string literal containing the name of the source file. The value of __FILE__ changes as the compiler processes include files that are part of your source program. It can be set with the `#line` directive.
__DATE__	A character string literal containing the date when the source file was compiled. The value of __DATE__ changes as the compiler processes any include files that are part of your source program. The date is in the form Mmm dd yyyy, where Mmm represents the month in an abbreviated form (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), dd represents the day of the month (if the day is less than 10, the first d is a blank character), and yyyy represents the year.
__STDC__	The integer 1 (one) indicates that the C compiler conforms to the ANSI standard. This macro is undefined if the language level is set to anything other than ANSI.
__TIME__	A character string literal containing the time when the source file was compiled. The value of __TIME__ changes as the compiler processes any included files that are part of your source program. The time is in the form hh:mm:ss, where hh represents the hour, mm represents the minutes, and ss represents the seconds. The time is always set to the system time.
__TIMESTAMP__	A character string literal containing the date and time when the source file was last modified. The value of __TIMESTAMP__ changes as the compiler process any include files that are part of your source program. The date and the time are in the form Day Mmm dd hh:mm:ss yyyy, where Day represents the day of the week. (Mon, Tue, Wed, Thu, Fri, Sat, or Sun), Mmm represents the month in an abbreviated form (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), dd represents the day of the month (if the day is less than 10, the first d is a blank character), hh represents the hour, mm represents the minutes, ss represents the seconds, and yyyy represents the year. The date and time are always set to the system date and time.

Table 77. AIX 5L specific predefined preprocessor macros

Predefined macro	Description
__64BIT__	Defined if the compiler is invoked to compile in 64-bit mode. This macro should not be user-defined or redefined.
_AIX32	Defined if the operating system is AIX Version 3.2 or higher.
_AIX41	Defined if the operating system is AIX Version 4.1 or higher.
_AIX43	Defined if the operating system is AIX Version 4.3 or higher.
__ANSI__	Allows only language constructs that conform to ANSI C standards. Defined using the `#pragma langlvl` directive or the -qlanglvl compiler option.
ARCH<arch>	Indicates that the compiler generates code to run on the family of processors denoted by <arch>. See the -qarch option in Section 8.1.7.14, “Target platform compiler options” on page 243.
_CHAR_SIGNED	Indicates that the default character type is signed. Defined when the -qchars=signed compiler option is in effect. See the -qchars compiler option for more information.
CHAR UNSIGNED	Indicates that the default character type is unsigned. Defined when the -qchars=unsigned compiler option is in effect. See the -qchars compiler option for more information.
__CLASSIC__	Macro defined when the classic language level is specified. Defined using the `#pragma langlvl` directive or the -qlanglvl compiler option.
__EXTENDED__	Allows additional language constructs provided by the C for AIX implementation. Defined using the `#pragma langlvl` directive or the -qlanglvl compiler option.
__FUNCTION__	Indicates the name of the function being compiled.
__HOS_AIX__	Indicates the host operating system is AIX.
__IBMC__	Macro contains the version number of the compiler, for example, __IBMC__=502. This macro should be used in new code.
__IBMSMP	Macro defined when the -qsmp compiler option is selected.
_ILP32	Defined if the compiler is using the 32-bit data model. This data model is used when compiling programs for the 32-bit mode. This macro should not be user-defined or redefined.
_LONG_LONG	Macro defined when the compiler is in a mode that permits long long int and unsigned long long int types.

Predefined macro	Description
_LONGDOUBLE 128	Sets the number of bits to use when representing the value of a long double. The available options are 64 and 128 bits.
_LP64	Defined if the compiler is using the 64-bit data model. This data model is used when compiling programs for 64-bit mode. This macro should not be user-defined or redefined.
__MATH__	Instructs the compiler to generate substitute code for calls to some math functions available in the standard C run-time libraries, if appropriate. The functions handled this way are defined as replacement text for macros that begin with two underscores (__) in the /usr/include/math.h header file.
_OPENMP	Macro defined when the -qsmp=omp compiler option is set to enable full compliance to the OpenMP API specification.
_POWER	Indicates the operating system is AIX Version 4.1 or higher.
__SAA__	Allows only language constructs that conform to the most recent level of the SAA C standards. Defined using the `#pragma langlvl` directive or the -qlanglvl compiler option.
__SAAL2__	Allows only language constructs that conform to the most recent level of the SAA Level 2 C standards.
__STR__	Instructs the compiler to generate substitute code for calls to some string functions available in the standard C run-time libraries, if appropriate. The functions handled this way are defined as replacement text for macros that begin with two underscores (__) in the /usr/include/string.h header file.
__THW_INTEL__	Indicates that the target hardware is an Intel processor.
__THW_RS6000__	Indicates that the target hardware is a RS/6000 processor.
__xlC__	A hexadecimal constant containing the version number of the compiler. The version is in the form: 0xVVRR where: VV represents the compiler version number. RR represents the compiler release number.

Predefined macro	Description
__XLC121__	Instructs the compiler to generate substitute code for calls to some new string and math functions. The functions handled this way are defined as replacement text for macros that begin with two underscores (__) in the following header files: /usr/include/string.h /usr/include/math.h /usr/include/stdlib.h /usr/include/stream.h

Note that you cannot change the values of the ANSI predefined macros. An example of this can be see in Figure 54, where we try to undefine the __LINE__ macro by using the -U flag to the `cc` command.

```
$ ls
bar.c  foo.c
$ cc -c -U"__LINE__" foo.c
"foo.c": 1506-085 (E) Predefined macro __LINE__ cannot be undefined.
$
```

Figure 54. Trying to undefine __LINE__

In Figure 55, we try to use the precompiler directive `#define` to redefine the __TIME__ macro.

```
$ ls
bar.c  foo.c
$ grep -n '#define' *
foo.c:8:#define __TIME__
$ cc -c  foo.c
"foo.c", line 8.9: 1506-188 (E) Reserved name __TIME__ cannot be defined
as a macro name.
$
```

Figure 55. Trying to redefine __TIME__ using #define

If you try to redefine or undefine the AIX 5L C compiler provided predefined preprocessor macros, you will either get a warning or an error, depending on whether you have used a `#pragma` or compiler option that defines that specific macro. An example can be seen in Figure 56 on page 248, where we have made a `#undef` and `#define` of _CHAR_SIGNED and _CHAR_UNSIGNED.

```
$ cc -c  foo.c -qchars=unsigned
"foo.c", line 22.8: 1506-313 (W) Compiler internal name _CHAR_SIGNED has been undefined as a
macro.
"foo.c", line 23.9: 1506-312 (W) Compiler internal name _CHAR_SIGNED has been defined as a
macro.
"foo.c", line 24.8: 1506-085 (E) Predefined macro _CHAR_UNSIGNED cannot be undefined.
"foo.c", line 25.9: 1506-188 (E) Reserved name _CHAR_UNSIGNED cannot be defined as a macro
name.
$ cc -c  foo.c -qchars=signed
"foo.c", line 22.8: 1506-085 (E) Predefined macro _CHAR_SIGNED cannot be undefined.
"foo.c", line 23.9: 1506-188 (E) Reserved name _CHAR_SIGNED cannot be defined as a macro
name.
"foo.c", line 24.8: 1506-313 (W) Compiler internal name _CHAR_UNSIGNED has been undefined as
a macro.
"foo.c", line 25.9: 1506-312 (W) Compiler internal name _CHAR_UNSIGNED has been defined as a
macro.
$
```

Figure 56. Changing -qchars to get warnings

One practical use of predefined preprocessor macros can be to display build information. If you look at the program shown in Figure 57 on page 249, and the output of the compiled program shown in Figure 58 on page 249, you can see how the macros are used to print out build information.

There are lot of other practical uses for predefined preprocessor macros. They can be to distinguish between:

- Different operating systems
- Different operating system versions
- Different hardware
- Different processors
- 64/32 bit

```
/*
 * This example tries to show how predefined precompiler macros
 * can be used in a program.
*/
#include <stdio.h>
#ifdef __TWH_RS6000__
  #define HARDWARE RS6000
#elif __THW_INTEL__
  #define HARDWARE INTEL
#endif
#ifdef __IBMC__
  #
#else
  #define __IBMC__ 0x0
#endif
#ifdef _AIX50
  #define AIXVER "AIX 5.0"
#elif _AIX43
  #define AIXVER "AIX 4.3"
#elif _AIX41
  #define AIXVER "AIX 4.1"
#elif _AIX32
  #define AIXVER "AIX 3.2"
#endif
#ifdef __64BIT__
  #define BITS 64
#else
  #define BITS 32
#endif

int main(void)
{
  printf("This program was build at %s on %s\n", __TIME__, __DATE__);
  printf("Using ver. %d of the C compiler on %s or higher.\n",__IBMC__,AIXVER);
  printf("This program is compiled for %d bits execution.\n",BITS);
}
```

Figure 57. A simple C program that uses #defines to determine build information

```
$ cc -q64 build.c -o build
$ ./build
This program was build at 19:14:00 on Mar 22 2001
Using ver. 500 of the C compiler on AIX 5.0 or higher.
This program is compiled for 64 bits execution.
$
```

Figure 58. Compiling and running the program in Figure 57

8.2 GNU GCC for AIX 5L

We have chosen not to compare the GCC compiler to the AIX 5L C compiler. This is partly because of the sheer number of options you can specify for the

GCC compiler, but mainly due to the fact that if you are using GCC on your source platform at the moment, then you probably will be using it on AIX 5L also.

The primary force of the GCC compiler, besides from it being freeware, is its great portability. But the price the GCC compiler pays is that it does not generate code that runs as fast as the native compiler for a particular platform (this is also true for the AIX 5L platform).

But in many cases, speed is not the key target, but having one development environment on many hardware platforms might be the prime motivation for choosing a compiler. For those situations, the GNU C Compiler might be the best choice, at least to start with.

In Appendix C.14, “GCC options specific for AIX 5L” on page 580, there is a list of the supported options for AIX 5L on Power and Itanium-based systems.

8.3 The C++ compiler

This chapter deals with the VisualAge C++ compiler for AIX 5L. Due to the fact that the C++ compiler on AIX 5L is a superset of the C compiler, this section only deals with the things that are different for the C++ compiler.

8.3.1 Introduction

Due to the fact that the subject of this redbook is about porting from another platform to AIX 5L, we will only be talking about the batch compiler part of VisualAge C++. If you wish to get further information or wish to utilize other features of the VisualAge C++ compiler, there is, in Section 4.11.1.2, “C++ compiler documentation” on page 105, links to the online documentation for VisualAge C++ compiler.

8.3.2 Types of input files

The VisualAge C++ compiler takes the same input files as the C compiler, with the addition of C++ source files.

C++ source files have several different suffixes, as we saw in Section 6.1.3, “Single suffix default inference rules” on page 159. The most common are:

- .C
- .cpp
- .cxx
- .cc

So one thing you might want to do is to consider renaming all your .cc, .cpp and .cxx files to .C, which is the suffix used for C++ files on AIX 5L. The VisualAge C++ compiler does recognize all the above file suffixes as C++ source files. This will also enable you to use the default inference rules of the `make` command. If you do want to keep the suffixes, you can always copy the default inference rules that deal with C++ source files (the .C ones) and include them in your makefiles after altering them to understand the suffix you are using.

If your C++ source files use another file name suffix, another option is to use the -+ compiler option, which informs the compiler that the file given as an argument is a C++ source file. This is done in the way listed here:

```
xlC -+ cplusplussroucefile.C++ -o cplusplusexe
```

8.3.3 VisualAge C++ compiler files and directories

This section should give you a feel of where the different parts of the VisualAge C++ compiler are installed. In Table 78 we have the directory structure of the C++ compiler. The C++ compiler is installed in the /usr/vacpp, directory. It uses the same configuration files as the C compiler.

Table 78. Directory structure of the VisualAge C++ compiler

Description	Directory
Configuration files for the C and C++ compiler	/etc/
Directory that holds the C++ compiler binaries	/usr/vacpp/bin/
Root directory for the C++ compiler	/usr/vacpp/
Header files (This depends on what you actually have installed on the machine.)	/usr/include/ /usr/vac/include/ /usr/vacpp/include
Library files	/usr/vacpp/lib/ /usr/vacpp/lib/<os version>
Message files	/usr/lib/nls/msg/$LANG/
Directory that holds the C compiler tools, such as the preprocessor, assembler and disassembler	/usr/vacpp/exe/

The programs in /usr/vacpp/exe/ are not intended to be used from the command line. Files used by the VisualAge C++ compiler are listed in Table 79.

Table 79. Files used by the VisualAge C++ compiler

Description	File
VisualAge C ++ README file, which contains important information not included in other documentation. Read this file before you use the compiler for the first time.	/usr/vacpp/README
VisualAge C++ front end.	/usr/vac/exe/xlCentry
Help file.	/usr/vac/exe/default_msg/vac.help
C preprocessor.	/usr/vac/exe/xlCcpp
Disassembler.	/usr/vac/exe/dis
Code generator.	/usr/vac/exe/xlCcode /usr/vac/exe/bolt
Driver programs.	/usr/vac/bin/xlc /usr/vac/bin/xlc128 /usr/vac/bin/xlc_r /usr/vac/bin/cc /usr/vac/bin/cc128 /usr/vac/bin/cc_r /usr/vac/bin/cleanpdf /usr/vac/bin/replaceCSET /usr/vac/bin/resetpdf /usr/vac/bin/restoreCSET /usr/vac/bin/showpdf
Profiling library.	/usr/vacpp/lib/profiled
Configuration files.	/etc/vac.cfg.<oslevel>
Link to actual used configuration file.	/etc/vac.cfg

Description	File
Links to /usr/vac/bin	/usr/bin/xlc /usr/bin/xlc128 /usr/bin/xlc_r /usr/bin/c89 /usr/bin/cc /usr/bin/cc128 /usr/bin/cc_r /usr/vacpp/bin/xlC128_r /usr/vacpp/bin/xlC128 /usr/vacpp/bin/xlC /usr/vacpp/bin/CC_r4 /usr/vacpp/bin/xlc_r7 /usr/vacpp/bin/xlc_r /usr/vacpp/bin/xlc128 /usr/vacpp/bin/xlc /usr/vacpp/bin/xlC_r7 /usr/vacpp/bin/xlC_r4 /usr/vacpp/bin/xlC_r /usr/vacpp/bin/xlC128_r7 /usr/vacpp/bin/xlC128_r4 /usr/vacpp/bin/showpdf /usr/vacpp/bin/resetpdf /usr/vacpp/bin/cleanpdf /usr/vacpp/bin/cc_r7 /usr/vacpp/bin/cc_r4 /usr/vacpp/bin/cc_r /usr/vacpp/bin/cc128 /usr/vacpp/bin/cc /usr/vacpp/bin/c89
Other executables used by the compiler	/bin/as /bin/ld /bin/strip

8.3.4 Command line arguments

There are several options that are specific to the C++ language environment.

8.3.4.1 The priority compiler option

The priority compiler option specifies the priority level for the initialization of static constructors. The syntax of the priority compiler option can be written like this:

```
-qpriority=number
```

The number is the initialization priority level assigned to the static constructors within a file, or the priority level of a shared or non-shared file or library. You can specify a priority level from -(2147483647 + 1) (highest priority) to +2147483647 (lowest priority).

For example, to compile the file myprogram.C to produce an object file myprogram.o so that objects within that file have an initialization priority of -200, enter:

```
xlC myprogram.C -c -qpriority=-200
```

All objects in the resulting object file will be given an initialization priority of -200, provided that the source file contains no `#pragma priority(number)` directives specifying a different priority level.

8.3.4.2 The staticinline compiler option

This option controls whether inline functions are treated as static or extern. By default, VisualAge C++ treats inline functions as extern. The syntax for the staticinline compiler option can be written like this:

```
-qstaticinline
```

or

```
-qnostaticinline
```

For example, using the -qstaticinline option causes function f in the following declaration to be treated as static, even though it is not explicitly declared as such.

```
inline void f(){/*...*/};
```

Using the default -qnostaticinline gives f external linkage.

8.3.4.3 The compiler twolink option

Minimizes the number of static constructors included from libraries. The syntax can be written like this:

```
-qtwolink
```

or

```
-qnotwolink
```

Normally, the compiler links in all static constructors defined anywhere in the object (.o) files and library (.a) files. The -qtwolink option makes link time take longer, but linking is compatible with older versions of C or C++ compilers. Before using -qtwolink, make sure that any .o files placed in an archive do not change the behavior of the program.

The default is -qnotwolink. All static constructors in .o files and object files are invoked. This generates larger executable files, but ensures that placing a .o file in a library does not change the behavior of a program.

8.3.5 Predefined preprocessor macros

Besides the predefined preprocessor macros defined in Section 8.1.8, “Predefined preprocessor macros” on page 243, there are some that are specific to the VisualAge C++ compiler. These are listed in Table 80.

Table 80. Specific predefined macro for C++

Predefined Macro	Description
__TEMPINC__	The macro __TEMPINC__ is defined in all compilation units in which automatic template generation is used.
__IBMCPP__	The version of the of the VisualAge C++ compiler. The value is 502 for VisualAge C++ Professional for AIX Version 5.0.2.
__cplusplus	Defined if the source is C++; otherwise, it is not defined.

8.4 Migrating to VisualAge C++ Version 5

The IBM C++ compiler for AIX 5L conforms to the ANSI C++ language specification. When migrating to AIX from other platforms (or from previous versions of AIX), it is possible that, in addition to moving from one operating system to another, you are migrating to ANSI C++ for the first time.

ANSI C++ introduced a number of changes and additions to the language that may not be 100 percent compatible with existing code. This sections details some of the more common problems that may occur.

8.4.1 New keywords

The C++ standard now defines the tokens bool, true, and false as keywords. When you migrate programs that define these keywords, you will encounter compilation errors. You can either remove your definitions, or use the -qnokeyword option for each of these keywords that you want to undefine for compatibility purposes. For example, to disable all three keywords, add the following option to the command line:

```
-qnokeyword=true|false|bool
```

8.4.2 Changes to digraphs in the C++ language

The C++ standard now defines and, bitor, or, xor, compl, bitand, and_eq, or_eq, xor_eq, not and not_eq as alternate tokens for &&, |, ||, ^, ~, &, &=, |=,

^=, ! and !=. If any of these alternate tokens are used as variable, function, or type names, then you can add -qnodigraph to the command line to suppress the parsing of these tokens as digraphs.

Note that the -qnokeyword option cannot be used to disable the digraph.

Chapter 9. AIX shared objects and libraries

Facilities for the creation and use of shared libraries are found on many operating systems. The AIX 5L operating system is no exception and provides a large number of useful tools to aid in the creation, development, testing, and debugging of shared libraries and applications that use them.

Developers porting code to AIX 5L running on the Power platform from other operating systems may be, at first, troubled by the different implementation methods that are available. Beginning with AIX Version 4.3, AIX contained shared library features that are broadly compatible with other UNIX operating systems. Previous versions of AIX did not contain all of these features.

Note that AIX 5L supports two hardware platforms, namely Power systems and Itanium-based systems. For various reasons, there are minor differences between the shared library implementation on each platform; however, these differences can be contained within the configuration options of the makefile system.

The AIX 5L operating system provides facilities for the creation and use of *dynamically bound* shared libraries. Dynamic binding allows external symbols referenced in user code and defined in a shared library to be resolved by the loader at run time.

The shared library code is not present in the executable image on disk. Shared code is loaded into memory once in the system shared library segment and shared by all processes that reference it. The advantages of shared libraries are:

- Less disk space is used because the shared library code is not included in the executable programs.
- Less memory is used because the shared library code is only loaded once.
- The time taken to start an application may be reduced because the shared library code may already be in memory.
- Performance of the application program may be improved because fewer page faults will be generated when the shared library code is already in memory. However, there is a performance cost in calls to shared library routines of one to eight machine instructions.

This chapter introduces the developer to shared libraries and their implementation in AIX 5L. At the time of writing, compilers have only been released for the IBM Power platform. Hence, this chapter is mainly concerned with the IBM Power platform, but Section 9.11, “Linker differences on Itanium-based systems” on page 299 discusses the main differences between the linkers on the two platforms.

9.1 Terminology

When discussing shared libraries, it is very important to understand the terminology used, since there are many terms with similar names but different meanings.

9.1.1 Static library

A static library is a collection of object files in a single ar format archive. The library can be used during the linking phase of creating an executable. The object files in the library that contain symbols referenced by the main application are extracted from the library and incorporated into the resulting executable file. The library is used only during the linking phase and is not relevant at run time. The executable file that is created is sufficient to run the program. This terminology is applicable to both Power and Itanium-based systems.

9.1.2 Shared library

Shared libraries and shared objects (normally called Dynamically Loaded Libraries, or DLLs in Windows terminology) are terms used to refer to object code components that are handled in a special way.

Shared libraries are used in two stages when creating an executable. At link time, the link editor (the `ld` command) searches the specified library to resolve all undefined symbols that are referenced in the main application code. If a shared library contains the referenced symbols, the loader section of the header of the created executable contains a reference to the shared library (see Section 8.1.3, “Types of input files” on page 224). Unlike using the static library, the object files containing the referenced symbols are not incorporated into the executable. Refer to Figure 59 on page 259 for a graphical representation. At run time, the system loader (the kernel component that starts new processes) reads the header information of the executable and attempts to locate and load any referenced shared libraries. Assuming all the referenced shared libraries are found, the executable can be started. This process is known as dynamic linking.

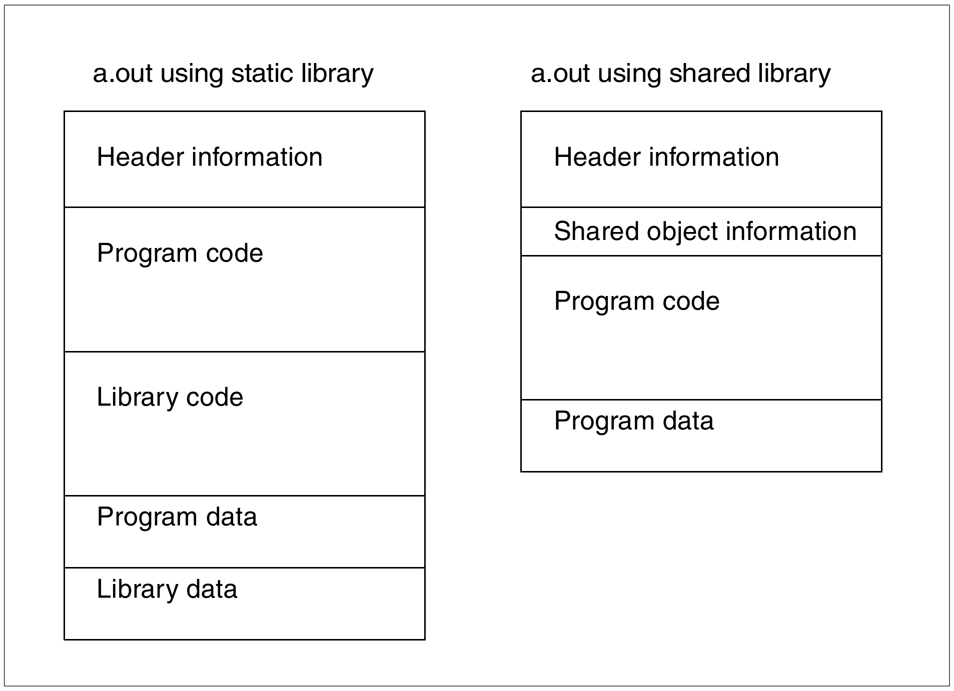


Figure 59. Executables created using static library and shared library

Figure 59 shows the difference between two executables created using the same main application code. One is created using a static version of the library, the other with a shared object version of the same library.

The object code for shared libraries that get loaded into system memory when starting an executable can then be shared by all subsequent executables that use the library. The benefit of this is that only one copy of the object code of a shared library is stored in system memory at any given time, with all the executing programs sharing the same copy. Thus, dynamic linking uses far less memory to run programs. Additionally, the executable files are much smaller, thus potentially also saving disk space.

The AIX 5L operating system supports dynamic linking. Developers moving code to AIX 5L often have problems, however, as the implementation specifics on the Power platform are slightly different from other platforms.

In the UNIX world, the terms shared library and shared object are generally used interchangeably. On the AIX system, there has historically been a distinct difference between the two terms.

Shared object	A shared object is a single object file that has the SRE (Shared REusable) bit set in the XCOFF header. A shared object normally has a name of the form, filename.o. In other words, it is a regular file with a .o (lower case O) extension to indicate it is an object file. The SRE bit indicates that the file is handled in a special way by the linker.
Shared library	On the Power platform, a shared library refers to an ar format archive library, where one or more of the members is a shared object. Note that the library can also contain normal, non-shared object files, which are handled in the normal way by the linker. A shared library normally has a name of the form, lib*name*.a. This allows the linker to search for libraries specified with the -lname option on the command line.

9.1.3 Itanium-based system differences

On Itanium-based systems, the term shared library and shared object are used interchangeably. The reason for this is that the system loader on AIX 5L for Itanium-based systems does not allow a static (ar format) library to contain shared objects.

Another difference between the Power and Itanium-based system implementation of AIX 5L concerns the format of the ar library. AIX 5L supports both 32-bit and 64-bit object code on both hardware platforms. On the Power platform, it is possible to have a single ar format archive library that contains both 32-bit and 64-bit objects. The linker (`ld`) command and the system loader (the part of the kernel that loads a new process) both understand that a library can contain both 32-bit and 64-bit objects. When processing an archive, they will only examine the appropriate type of objects. For example, when the `ld` command is being used to create a 64-bit user executable, it will search any specified libraries for the required 64-bit objects to complete the symbol resolution process. This is because neither Itanium or Power systems support mixed-mode processes (a binary consisting of both 32-bit and 64-bit object code).

The linker `ld` command and system loader on Itanium-based systems do not support mixed object libraries. For this reason, there are two versions of each system library: a 32-bit version and a 64-bit version. On the Power platform, there is only one instance of each system library, because the library can contain both 32-bit and 64-bit objects.

In addition to these format differences, the default action of the linker/loader combination on AIX 5L for Itanium-based systems is to use run-time linking equivalent to that performed on Power systems when the -brtl option is specified.

As an illustration of these library differences, the following section compares the different components of the C library (libc.a) on both Power and Itanium-based systems.

9.1.3.1 The Power standard C library

The C library on Power systems is /usr/ccs/lib/libc.a. The symbolic link /usr/lib/libc.a points to this file. The library contains both 32-bit and 64-bit objects. Some of the objects are static, and some are shared. The following command lists both 32-bit and 64-bit objects in the C library:

```
# ar -t -v -X32_64 /usr/lib/libc.a
rw-rw----  2715/300     1679 Aug 05 13:17 1998 frexp.o
rw-rw----  2715/300      750 Apr 22 09:15 1998 itrunc.o
rw-rw----  2715/300     1943 Aug 05 13:17 1998 ldexp.o
rw-rw----  2715/300     1575 Aug 05 13:17 1998 modf.o
rw-rw----  2715/300      909 Apr 22 08:40 1998 logb.o
rw-rw----  2715/300     2469 Aug 05 12:48 1998 scalb.o
rw-rw----  2715/300      330 Apr 22 08:40 1998 finite.o
rw-rw----  2715/300      744 Apr 22 09:16 1998 uitrunc.o
rw-rw----  2715/300      768 Apr 22 08:32 1998 _itrunc.o
r-xr-xr-x     2/2       1828 Nov 27 14:36 2000 frexp_64.o
r-xr-xr-x     2/2        891 Nov 27 14:36 2000 itrunc_64.o
r-xr-xr-x     2/2       2174 Nov 27 14:36 2000 ldexp_64.o
r-xr-xr-x     2/2       1838 Nov 27 14:36 2000 modf_64.o
r-xr-xr-x     2/2       1058 Nov 27 14:36 2000 logb_64.o
r-xr-xr-x     2/2     2774262 Nov 27 14:36 2000 shr.o
r-xr-xr-x     2/2     3153840 Nov 27 14:36 2000 shr_64.o
. . . . . .
. . . . . .
. . . . . .
```

The files with 64 in the name are 64-bit objects. There is a 32-bit and 64-bit version of most files. The files shr.o and shr_64.o are the shared object components of the C library.

9.1.3.2 The Itanium standard C library

There are two C libraries on Itanium-based systems. One is for 32-bit processes and the other is for 64-bit processes. This is because AIX 5L on Itanium-based systems does not support mixed object libraries. The 32-bit system libraries are in the directory /usr/lib/ia64l32, and the 64-bit versions are in /usr/lib/ia64l64:

```
itsoia64:/>ls -l /usr/lib/ia64l??/libc.so
-r-xr-xr-x   3 bin      bin         4161368 Mar 07 12:21 /usr/lib/ia64l32/libc.so
-r-xr-xr-x   3 bin      bin         4320600 Mar 07 12:21 /usr/lib/ia64l64/libc.so
```

The C library itself is named libc.so and is a single shared object rather than an ar format archive, as on the Power platform.

AIX Version 4.2.1 introduced support for a new type of shared object, commonly found on other UNIX-based systems, such as Solaris and HP-UX. Shared files of the new format generally have a name of the form, lib*name*.so. Although the name incorporates the term lib, the file is, in fact, a shared object (as indicated by the .so file name extension) rather than a shared library, since it is a single object file rather than an ar format archive. The benefit of this type of shared object is that, in common with a true shared library, it can be specified on the compiler or linker command line with the -l*name* option and searched for with the -L*directory* option when the -brtl option is being used.

The system libraries on AIX 5L for Itanium-based systems use this format, although the -brtl option is not required.

In addition to the use of shared libraries and shared objects with the `compile` and `link` commands, a program may choose to explicitly control their use with the dlopen() family of subroutines.

9.2 Creating a shared library on Power systems

The method used to create a shared library depends on the type you wish to create.

9.2.1 Traditional AIX shared object

A traditional AIX shared object is a single object file created by a call to the linker (`ld`) command. Normally, the shared object is created from multiple object files that are linked together; however, it is also possible to create a shared object from a single object file. Although the linker is the component that actually does the work, it is normal to create the shared object using the compiler command line since the compiler, in turn, calls the linker once it has performed any processing it is capable of. The benefit of using this method to create the shared object is that default linker options are automatically used and do not need to be specified on the command line.

Creating a traditional AIX shared object normally involves the use of an export file. An export file is a text file containing a list of symbols. It is used to control which symbols are visible outside the shared object. The symbols not specified in the export file are only visible to other routines within the shared object. The use of export files allows a developer to create a shared object that has a well defined interface. Only the symbols listed in the export file can be referenced by executables and other shared objects that are linked with the object. In addition, creating a shared object may involve the use of an import file. An import file is a text file that lists the names of symbols that the shared object may reference. It allows the object to be created without the source of those symbols being available. This may be required in the situation where two shared objects have dependencies on each others symbols. Export files are normally identified by using a .exp extension to the file name. When the run-time linker (discussed in Section 9.5, “Run-time linking” on page 277) is not used, all symbols must be accounted for when the module is linked. The undefined symbols must be listed in the module’s import list or be deferred. Symbols are deferred if they are listed as being defined by #! in the import list.

If you are creating a shared object and want all symbols to be exported, then you do not need to use an export file. You can use the -bexpall linker option, which will automatically export all global symbols (except imported symbols, unreferenced symbols defined in archive members, and symbols beginning with an underscore). Additional symbols may be exported by listing them in an export list.

If the shared object supplies symbols that are used by another shared object, then you still have to create an exports file, as this is used as an import file when creating the dependent shared object.

9.2.1.1 Single shared object

The scenario described in this section for creating a shared object uses the following source code files:

The file source1.c is as follows:

```
/* source1.c : First shared library source */
void private(void)
{
   printf(“private\n”);
}
int addtot(int a , int b)
{
   int c;
   c = a+b;
   return c;
}
```

The file source2.c is as follows:

```
/* source2.c : Second shared library source */
#include <stdio.h>
int disptot(int a)
{
   printf("The total is : %d \n",a);
}
```

The process of creating the shared object is as follows:

1. Create the object files that will be combined together to create the shared object. This is achieved using the -c option of the compiler. For example:

   ```
   cc -c source1.c
   cc -c source2.c
   ```

2. Create an export file that lists the symbol names that should be visible outside the shared object. In this example, the symbols addtot and disptot are the names of the functions that will be called by the main application. The symbol names can also include variable names in addition to function names. The libadd.exp export file is as follows:

   ```
   #!
   addtot
   disptot
   ```

3. Create the shared object with the following command:

   ```
   cc -o shrobj.o source1.o source2.o -bE:libadd.exp -bM:SRE -bnoentry
   ```

   The -bE:libadd.exp option uses the file libadd.exp as an export file that lists the names of the symbols that should be exported. The -bM:SRE flag marks the resultant object file, shrobj.o, as a shared reusable object. The -bnoentry flag indicates that there is no entry point (main function) in the object file.

The `dump` command can be used to list the symbols that are exported (and imported) by the shared object. For example:

```
# dump -Tv shrobj.o

shrobj.o:

                        ***Loader Section***

                        ***Loader Symbol Table Information***
[Index]      Value      Scn     IMEX Sclass   Type           IMPid Name

[0]     0x00000000    undef      IMP     DS EXTref   libc.a(shr.o) printf
[1]     0x2000020c    .data      EXP     DS SECdef        [noIMid] addtot
```

```
[2]     0x20000218    .data      EXP     DS SECdef        [noIMid] disptot
```

The important fields to examine are the IMEX, IMPid, and Name entries. A value of EXP in the IMEX field indicates that this symbol is being exported from the object. In this case, the Name field gives the name of the symbol being exported, and the IMPid field is not used.

A value of IMP in the IMEX field means that the symbol listed in the Name field is being imported into the object. In this case, the IMPid indicates the target shared object that the symbol will be imported from. In the case of a shared object that is contained in an ar format library, both the library name and object name will be displayed. In the example shown above, the symbol printf is being imported from the shared object shr.o, which is contained in the libc.a archive library.

9.2.1.2 Interdependent shared objects

The process for creating interdependent shared objects is similar to the process of creating a single shared object but requires the use of an import file. Suppose there are two shared objects, shr1.o and shr2.o, and each references symbols in the other. When creating the first shared object (shr1.o), the second shared object may not exist. This means that when the command to create the first shared object is executed, there will be unresolved symbols since, at this point, the second shared object does not exist. This problem is overcome with the use of an import file. An import file is similar to the export file used when creating a shared object. In fact, in most cases, it is possible to use the same file for both purposes.

Consider the following files for use in this example scenario:

The file source1.c is as follows:

```
/* source1.c : First shared library source */
int function1(int a)
{
   int c;
   c = a + function2(a);
   return c;
}

int function3(int a)
{
   int c;
   c = a / 2;
   return c;
}
```

The file source2.c is as follows:

```
/* source2.c : Second shared library source */
int function2(int a)
{
   int c;
   c = function3(a + 5);
   return c;
}
```

In this example, each source file needs to be made into a separate, shared object. Note that the resulting shared objects are interdependent, because:

- Subroutine function1 in source1.c calls function2 in source2.c.
- Subroutine function2 in source2.c calls function3 in source1.c.

As with the simple example, each shared object requires an export file to define which symbols will be exported. With a slight modification, the export file for each shared object can also be used as the import file for other shared objects that use the exported symbols. The slight change does not affect the file when used as an export file. The modification is to add the name of the library and shared object that contains the symbols. In the example, the export file (libone.exp) for the first shared object is:

```
#!libone.a(shr1.o)
function1
function3
```

The export file (libtwo.exp) for the second shared object is:

```
#!libtwo.a(shr2.o)
function2
```

When used as an export file, the line starting with the #! symbol sequence is ignored. When used as an import file, the information following the #! sequence details the location of the symbols contained in the file. The format of the entry is libraryname(membername). The libraryname component can be just the name of the library (as it is in the example), or it may include a relative or absolute path name component, for example:

```
#!/data/lib/libone.a(shr1.o)
```

Any path name component listed is used when attempting to load the shared object at run time to resolve the symbols.

The commands used to create the shared objects and create the libraries are as follows:

```
cc -c source1.c
cc -o shr1.o source1.o -bE:libone.exp -bI:libtwo.exp -bM:SRE -bnoentry
ar rv libone.a shr1.o
cc -c source2.c
cc -o shr2.o source2.o -bE:libtwo.exp -bI:libone.exp -bM:SRE -bnoentry
ar rv libtwo.a shr2.o
```

Note the use of the file libone.exp as an export file when creating the first shared library and as an import file when creating the second. If the file is not used when creating the second shared library, the creation of the shared object will fail with an error message complaining of unresolved symbols:

```
cc -o shr2.o source2.o -bE:libtwo.exp -bM:SRE -bnoentry
ld: 0711-317 ERROR: Undefined symbol: .function3
ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more
information.
```

A single import file can be used to list symbols that are imported from different modules. The import file is just a concatenation of the individual export files for each of the shared objects. Using the example import files shown above, suppose that a new shared object, libthree.a, was to be created, and it imports symbols from both libone.a and libtwo.a. The import file used to create the new shared object might be as follows:

```
#!libone.a(shr1.o)
function1
function3
* a comment line starts with the asterix symbol
* blank lines are ignored

#!libtwo.a(shr2.o)
function2
```

As the example illustrates, although it is possible to create interdependent shared objects, from a design and implementation point of view, it is much simpler to create shared objects that are as self-contained as possible.

9.2.2 New style shared object

Creating a new style shared object (lib*name*.so) does not require the use of export files; however, by default, all symbols are visible to executables that are linked with the object.

9.2.2.1 Single shared object

Using the same source code as used in Section 9.2.1.1, “Single shared object” on page 263, the following command is used to create a new style shared object:

```
cc -G -o libsimple.so source1.o source2.o
```

Note that the -G option implicitly enables a number of other default linker options, including one that exports all symbols. This makes things simple when creating the shared object, since you do not need to maintain a file listing the symbols you want to be exported. The effect of this can be seen in the output of the `dump` command when used on the resulting shared object:

```
# dump -Tv libsimple.so

libsimple.so:

                        ***Loader Section***

                        ***Loader Symbol Table Information***
[Index]      Value      Scn     IMEX Sclass   Type           IMPid Name

[0]     0x00000000    undef      IMP     DS EXTref   libc.a(shr.o) printf
[1]     0x20000204    .data      EXP     DS SECdef        [noIMid] addtot
[2]     0x20000210    .data      EXP     DS SECdef        [noIMid] privatefn
[3]     0x2000021c    .data      EXP     DS SECdef        [noIMid] disptot
```

Although the manual pages for the compilers state that the -G option is passed directly to the linker, the compiler itself does, in fact, perform additional processing. This can be detected, because replacing `cc` with `ld` in the example shown above results in an error:

```
ld -G -o libsimple.so source1.o source2.o
ld: 0711-327 WARNING: Entry point not found: __start
ld: 0711-244 ERROR: No csects or exported symbols have been saved.
```

Even resolving the warning message about the entry point by using the -bnoentry linker option does not solve the problem. There is still a warning that no exported symbols have been saved. Essentially, this means the shared object has not exported any symbols.

The reason the command works when the compiler is invoked with the -G option can be seen when we additionally use the -v option to get more information about what the compiler is actually doing:

```
cc -v -G -o libsimple.so source1.o source2.o
exec: /usr/vac/bin/CreateExportList(/usr/vac/bin/CreateExportList,
/tmp/xlcSEMY4Qie,-f,/tmp/xlcSFMY4Qid,NULL)
exec: /bin/ld(ld,-bM:SRE,-bnoentry,-bpT:0x10000000,-bpD:0x20000000,
-olibsimple.so,source1.o,source2.o,-lc,-bE:/tmp/xlcSEMY4Qie,NULL)
unlink: /tmp/xlcW0MY4Qia
unlink: /tmp/xlcW1MY4Qib
unlink: /tmp/xlcW2MY4Qic
unlink: /tmp/xlcSEMY4Qie
unlink: /tmp/xlcSFMY4Qid
```

The important thing to notice is that the compiler is using a shell script called CreateExportList to create an export list file on the fly for the specified input files.

9.2.2.2 Creating an export list

You can use the /usr/vac/bin/CreateExportList shell script supplied with the C for AIX Version 5 compiler to automatically generate the symbols that should be included in an export list. It can save a considerable amount of time if you want to use the traditional AIX method for creating shared objects as described in Section 9.2.1, “Traditional AIX shared object” on page 262, or if you want to use an export list in conjunction with the -G option to create a new style shared object that does not export all symbols.

The simplest way to use the command is as follows:

1. Compile all of the source files that will be included in the shared object.
2. Create a single file that lists the names of all of the object files that will be included in the shared object. For example, create a file called objectlist that contains the following lines:

   ```
   source1.o
   source2.o
   ```

3. Invoke the `CreateExportList` command as follows:

   ```
   /usr/vac/bin/CreateExportList exportfile -f objectlist
   ```

   where exportfile is the name of the export file you want to create, and objectlist is the file that contains the list of object file names.
4. Edit the resulting export file to include the #!path name (member) line at the start.
5. Edit the resulting export file to remove the symbol names you wish to keep private within the shared object.

9.2.2.3 Interdependent shared objects

The creation of interdependent shared objects using the lib*name*.so style requires the use of import files so that the linker can resolve the externally referenced symbols.

If using an export file for a new style shared object as an import file when creating another shared object, the location specified does not need the (member) entry since the file itself is the shared object. Using the example described in Section 9.2.2.1, “Single shared object” on page 267, the export file produced would have the following line inserted as the first line in the file:

```
#!libsimple.so
```

9.2.3 Importing symbols from the main program

When creating either traditional or new style shared objects, it is possible for the object to resolve a symbol that is provided in the main program rather than a shared object. There are two steps required to ensure that this works correctly.

The first step is to use an import file when creating the shared object that lists the symbols to be imported from the main routine. The symbols should be listed under the module name as follows:

```
#!.
```

The special module name of . (dot) indicates that the symbols will be imported from the main program. The status of the symbols in the shared object can be checked using the `dump -Tv` command, as described in Section 9.6.3.2, "The dump -Tv command" on page 285.

Link the application using the shared objects as normal. The linker will automatically detect that the shared objects import symbols from the main routine and will automatically export them if they exist. If a shared object tries to import a symbol that does not exist in the main routine, then the link stage will fail.

9.2.4 Initialization and termination routines

Optional shared object initialization and termination routines can be specified when creating the shared object. You can use one or the other, or both. The routines may be useful for initializing dynamic data structures or reading configuration information. The initialization routines are called by the program startup code and are performed before the application main routine is started. Termination routines are called when the program makes a graceful exit. They will not be called if the program exits due to receipt of a signal.

The -binitfini linker option is used to specify the names of the routines along with a priority number. The priority is used to indicate the order that the routines should be called in when multiple shared objects with initialization or termination routines are used.

9.3 Creating a shared object on Itanium-based systems

Shared objects are created on Itanium-based systems by using the `ld` command with the -G option. This is similar to the procedure used on some other UNIX-based platforms.

For example, to make the shared object libone.so from the source files source1.c and source2.c, use:

```
cc -g -c source1.c
cc -g -c source2.c
ld -G -o libone.so source1.o source2.o
```

Since AIX 5L on Itanium-based systems uses run-time linking by default, there is no need to use import files or export files for symbol resolution, or when building interdependent objects. This is much easier than on Power systems; however, the system loader has to do more work when loading an application, and if there are problems with missing symbols, it is not as easy to diagnose the exact problem.

9.4 Using a shared library

Once you have created the required shared libraries, you can then proceed to use them when linking applications. There are a number of linker options that affect the way in which the shared libraries are used.

The most important point to remember about using shared libraries is that the way the application is linked will determine how the shared libraries will be searched for at run time.

9.4.1 On the compile line

When using shared libraries to create an executable, there are a number of methods that can be used to specify the library on the command line. The method used will depend on the type of shared object being used.

As far as the linker is concerned, there are three types of shared objects that it can handle:

- An archive library that contains object files with the SRE bit set. Note that this is only available on Power systems.
- A new style shared object of the form lib*name*.so.
- An individual object file with the SRE bit set, for example, shr1.o.

In all cases, the shared object can be specified directly on the command line using either an absolute or relative path name. If the shared object is in the same directory as the current working directory, then no path component needs to be specified, since the current directory is searched by default.

If the shared object is a single object file, then the absolute or relative path name is the only way to include it on the command line.

If the shared object is part of an archive library, then the -l and -L linker options can be used to search for the library. If the shared object is a new style shared object, then the -brtl linker option must be used. This enables the run-time linker, described in Section 9.5, "Run-time linking" on page 277, and also allows these shared objects to be specified on the command line using the -l and -L options. If you want to use the new style shared object naming conventions, but do not want to use run-time linking, then specify the -brtl and -bnortllib options when linking the main application. This will mean that you must build the new style shared objects using export and import files, if required. You should use the compiler with the -G option to create the shared objects for Power platforms, not the `ld -G` method described in Section 9.5, "Run-time linking" on page 277.

The -l option is used to specify the name of the library without the .a or .so extension and without the lib prefix. For example, the shared objects libone.a and libtwo.so, would be specified on the command line as `-lone -ltwo`.

The -L option is used to specify a directory that should be searched for in the libraries specified with the -l option. The /usr/lib and /lib directories are automatically added to the end of the list of directories to be searched. The list of directories specified with the -L option (along with the default /usr/lib and /lib entries) is included in the header section of the resulting executable. This path is used to search for the directories at run time. Refer to Section 9.6.3.1, "The dump -H command" on page 283 for details on how the use of path names, and the -L option, can have an impact on how the system loader searches for the shared objects at run time on Power platforms.

If your application development directory structure does not match the directory structure used when your application is installed in a production environment, then, potentially, you need to adjust the arguments used with the linker to ensure that the resulting executables have the desired library search path.

For example, consider an application that has a development source code tree, as shown in Figure 60 on page 273.

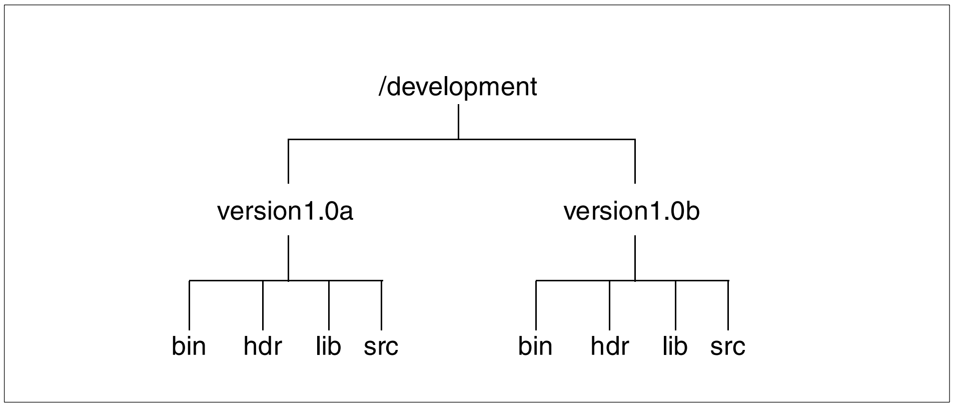


Figure 60. Sample development directory structure

Consider the application file, main.c, being compiled and linked in the directory /development/version1.0b/src and using shared libraries stored in the directory /development/version1.0b/lib. There are a number of options that can be used to specify the libraries, depending on how the resulting executable will be deployed.

When the application is installed in a production environment, for example, after being installed on a customer machine, the directory structure may be different. The method to use when compiling the executables will depend on the degree of freedom the customer is permitted when installing the application. For example, some products specify that the executables and libraries must be installed in a specific directory, such as /opt/productname. Some products allow the binaries and libraries to be installed in any directory structure.

If the libraries for the product will be installed in a specific directory, then you can either:

- Create the shared libraries and then copy them to the same directory structure to be used when the product is installed in a production environment. In this case, you use the -L option to find the shared libraries. For example:

  ```
  cc -o ../bin/app1 main.c -L/product/lib -lone
  ```

- Create the shared libraries, but leave them in the development directory structure. When compiling the applications, use absolute path names to specify the shared libraries along with the -bnoipath linker option to prevent the path name being included in the header section of the final executable. At the same time, use the -L option to specify the directory where the libraries will exist on a production system. For example:

```
cc -o ../bin/app1 main.c -bnoipath ../lib/libone.a -L/product/lib
```

If your product allows the executables and libraries to be installed in any directory structure, then you need to use the LIBPATH environment variable to search for shared objects. AIX 5L for Itanium-based systems (along with other UNIX-based systems) uses the LD_LIBRARY_PATH variable for the same purpose.

The order of libraries and objects specified on the command line is not important on the Power platform unless run-time linking is being used. See Section 9.5, "Run-time linking" on page 277 for more information. The ordering is important on Itanium-based systems, due to differences in the linker.

9.4.2 Searching at run time

The LIBPATH or LD_LIBRARY_PATH environment variable is only needed when shared libraries exist in a different directory than that specified in the header section of the executable. The variable is a colon separated list of directory names. If it is set, the directories specified in the environment variable are searched for the required shared objects before the list of directories specified in the header section of the executable. The exception to this case is when a user other than root is attempting to run a setuid or setgid executable. In this case, only the directories listed in the header section of the executable are searched; the environment variable is ignored, even if set.

If a relative or absolute path name is used to specify a shared object when the application is compiled, and the -bnoipath option is not specified, then the system loader will only look for the shared object using the exact path name specified at link time for that object. Even if a shared object with the same name exists in a directory searched as part of the LIBPATH or INDEX 0 path included in the header section, it will be ignored.

If a shared object can not be found by the system loader when trying to start an executable, an error message similar to the following will be seen on Power platforms:

```
exec(): 0509-036 Cannot load program ex1 because of the following errors:
      0509-022 Cannot load library libone.so.
      0509-026 System error: A file or directory in the path name does not
exist.
```

On Itanium-based systems, the message will be similar to the following:

```
dynamic linker: ./example: could not find or could not open 'libone.so'
Killed
```

The missing objects will be listed with 0509-022 error messages. Use the `find` command to search the system for the missing shared objects. If the object is found, try setting the LIBPATH or LD_LIBRARY_PATH environment variable to include the directory that contains the shared object and restart the application. Also, ensure that the object or library has read permission for the user trying to start the application.

A similar error message is produced when the system loader finds the specified shared objects, but not all of the required symbols can be resolved. This can happen when an incompatible version of a shared object is used with an executable. The error message is similar to the following on Power platforms:

```
exec(): 0509-036 Cannot load program ./ex1 because of the following errors:
      0509-023 Symbol func1 in ex1 is not defined.
      0509-026 System error: Cannot run a file that does not have a valid
format.
```

On Itanium-based systems, the output is similar to the following:

```
itsoia64:/home/richard/new>./example
address of main is 0x100005e0
address of mainfdesc struct is 0x20200820
dynamic linker: ./example: relocation error: symbol not found 'function1';
referenced from './example'
Killed
```

Note that because of the ‘lazy’ run-time linking, the application manages to partially run before encountering the unresolved symbol. Compare this with the Power platform, where the application will not even start.

On the Power platform, the unresolved symbols are listed in the 0509-023 message lines. Note the name of the missing symbol, and use the `dump -Tv` command to determine which shared object the executable expects to resolve the symbol from. For example:

```
# dump -Tv ex1 | grep func1
[4]     0x00000000    undef      IMP     DS EXTref libone.a(shr1.o) func1
```

This indicates that the executable is expecting to resolve the symbol func1 from the shared object shr1.o which is an archive member of libone.a. This information can help you start the problem determination process.

9.4.3 Shared or non-shared

AIX 5L supports the use of the -bdynamic and -bstatic linker options to determine how a shared object should be treated by the linker. On Itanium-based systems, the options are -Bdynamic and -Bstatic.

These options are toggles and can be used repeatedly in the same link line. When dynamic is in effect, shared objects are used in the usual way. If you use the static option, remember to specify dynamic as the last option on the link line to ensure that the system libraries are treated as shared objects by the linker. If this is not done, and the system libraries are treated as normal archive libraries, the executable produced will be larger than normal. In addition, it will have the disadvantage that it may not work on future versions of AIX because it is hardcoded with a specific version of system libraries.

When the static option is in effect, shared objects are treated as regular files. On the Power platform, when -brtl is specified, and -bdynamic is in effect, the -l flag will search for files ending in .so, as well as those ending in .a. Refer to the following example:

```
cc -o main.o -bstatic -Lnewpath -lx -bdynamic
```

In this example, libx.a is treated as a regular archive file, even if it contains shared objects. The -bdynamic ensures that the system libraries, such as libc.a, are processed as shared objects:

```
cc -o main main.o -brtl -Lpath1 -Lpath2 -lx
```

Search for the object specified by -lx in the following order:

1. path1/libx.so
2. path1/libx.a
3. path2/libx.so
4. path2/libx.a

9.4.4 Lazy loading

AIX 5L on Power supports the use of the -blazy option to implement lazy loading. Lazy loading is a mechanism for deferring the loading of modules until one of its functions is required to be executed. Lazy loading is the default on Itanium-based systems. By default, the system loader automatically loads all of the module's dependants at the same time. By linking a module with the -blazy option, the module is loaded only when a function within it is called for the first time. Note that lazy loading works only if the run-time linker is not enabled. Also, only the modules referenced for their function can be lazy loaded.

9.5 Run-time linking

AIX 5L for Itanium-based systems uses run-time linking by default. This section applies to Power systems only.

As shown in the examples above, references to the symbols in the shared objects are, generally, bound at link time. That is, the output module associates an imported symbol with its definition in a specific object. The source of the definition can be seen by using the `dump -Tv` command on the executable or shared object. Refer to Section 9.6.3, “The dump command” on page 283 for more details.

At load time, the definition in the specified shared object is used even if other shared objects export the same symbol.

Programs can be modified to use the run-time linker, therefore allowing some symbols to be rebound at load time. To create a program that uses the run-time linker, link the program with the -brtl option. The way that shared modules are linked affects the rebinding of symbols.

To build shared objects enabled for run-time linking, use the -G flag and build the shared object with the `ld` command rather than the compiler `cc`, `xlc`, or `xlC` commands. The -G linker option enables the combination of options described in Table 81.

Table 81. The -G option

Option	Description
-berok	Enables creation of the object file, even if there are unresolved references.
-brtl	Enables run-time linking. All shared objects listed on the command line (those that are not part of an archive member) are listed in the output file. The system loader loads all such shared modules when the program runs, and the symbols exported by these shared objects may be used by the run-time linker.
-bsymbolic	Assigns this attribute to most symbols exported without an explicit attribute.
-bnortllib	Removes a reference to the run-time linker libraries. This means that the module built with the -G option (which contains the -bnortllib option) will be enabled for run-time linking, but the reference to the run-time linker libraries will be removed. Note that the run-time libraries should be referenced to link the main executable only.
-bnoautoexp	Prevents automatic exportation of any symbol.

Option	Description
-bM:SRE	Builds this module to be shared and reusable.

The function of the -G option to the `compiler` command is very similar in function to the -G option to the linker (`ld`) command, but there is a very subtle, yet important, difference when it comes to creating shared objects for use with run-time linking.

The important difference is the way the two options impact the handling of unresolved symbols. The following source code files will be used to demonstrate the difference.

File source1.c is used to make libone.so. The source code is as follows:

```
/* source1.c - demo of difference between cc -G and ld -G */
#include <stdio.h>
void function1(int a)
{
   printf("In function1\n");
   function2(a);
}
```

File source2.c is used to make libtwo.so. The source code is as follows:

```
/* source2.c - demo of difference between cc -G and ld -G */
#include <stdio.h>
void function2(int a)
{
   printf("In function2\n");
}
```

If the `compiler` command is used to create libone.so, it initially fails with an error message complaining about the unresolved symbol function2:

```
cc -G -o libone.so source1.c
ld: 0711-317 ERROR: Undefined symbol: .function2
ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more
information.
```

We can solve this immediate problem in one of two ways. We can supply an import file that resolves the symbol function2 to the shared object libtwo.so. However, if we do this, the shared object, libone.so, will be created with a reference to libtwo.so in the header section. This means we have resolved the symbol function2 at link time, which is not what we want. Alternatively, we can add the -berok option to the command line, which allows errors in the output file. If we do this, then the symbol function2 is unresolved at link time, which is what we want. We can then create libtwo.so, and then link both libraries with the following main.c program:

```
/* main.c - demonstrate how cc -G differs from ld -G */
int main(int argc, char ** argv)
{
   function1(45);
}
```

using the following command:

```
cc -o example main.c -brtl -L‘pwd‘ libone.so libtwo.so
```

Note the use of the -brtl option, which is required to enable the run-time linker. If we try and run the example program, an error message is produced:

```
# ./example
in function 1
Segmentation fault(coredump)
```

It can be seen from the output that the program has managed to start and get as far as the printf statement in function1. It then experiences a fatal error when trying to call function2. If we look at the header information for libone.so with the `dump -Tv` command, we can check the status of the symbols:

```
# dump -Tv libone.so

libone.so:

                        ***Loader Section***

                        ***Loader Symbol Table Information***
[Index]      Value      Scn     IMEX Sclass   Type           IMPid Name

[0]      0x00000000    undef      IMP     DS EXTref   libc.a(shr.o) printf
[1]      0x200001e8    .data      EXP     DS SECdef        [noIMid] function1
[2]      0x00000000    undef      IMP     DS EXTref        [noIMid] function2
```

It can be seen that the symbol function2 is marked as undef, which is what we expect. However, the problem is that the IMPid is marked as [noIMid], which means that the shared object does not know where to resolve the symbol function2. If we use the `ld` command to create the shared object instead of the compiler, then the result is slightly different. Create the shared object with the following commands:

```
cc -c source1.c
ld -G -o libone.so source1.o -bnoentry -bexapall -lc
```

The -bnoentry and -bexpall options are described previously. The -lc option is required to link the C library to resolve the printf function. If we look at the symbol information in the header section with the `dump -Tv` command:

```
# dump -Tv libone.so

libone.so:

                        ***Loader Section***

                        ***Loader Symbol Table Information***
[Index]      Value      Scn     IMEX Sclass   Type           IMPid Name

[0]     0x00000000    undef      IMP     DS EXTref   libc.a(shr.o) printf
[1]     0x00000008    .data      EXP     DS SECdef        [noIMid] function1
[2]     0x00000000    undef      IMP     DS EXTref              .. function2
```

the difference is the IMPid for the symbol function2. The shared object now thinks it will resolve the symbol from the special module called .. (dot dot). This indicates that the symbol will be resolved by the run-time linker. If we create libtwo.so using the same method, then the example program works correctly.

The run-time linker is called by the program startup code before entering the application's main routine.

When using run-time linking, the order of specifying libraries and objects on the command line is important. This is because the list of libraries and objects will be searched in sequence to resolve symbols that are imported from the special .. module. In addition, all of the shared objects specified on the command line will be included in the header section of the resulting executable. Using the example described above, the main program only calls the routine function1, which is in libone.so. Using the traditional style AIX link time symbol resolution, this would mean that the resulting executable would only reference libone.so in the header section. If this were the case, when the run-time linker is called, the shared object libtwo.so would not be present, and so the symbol resolution of function2, which is called from function1, would fail.

Another advantage of using run-time linking is that developers do not need to maintain a list of module interdependencies and import/export lists. By using the -bexpall option, all shared objects can export all symbols, and the run-time linker can be used to resolve the inter-module dependencies.

9.5.1 Rebinding system defined symbols

The shared libraries shipped with the AIX operating system are not enabled for run-time linking, but they can be enabled by using the `rtl_enable` command. For example, if a program defines its own version of the malloc routine, and wants to run in such a way that the routines in the libc.a shared objects also use the user defined version of malloc, then a new instance of libc.a must be first created. This can be done as follows:

```
rtl_enable -o /usr/local/lib/libc.a /lib/libc.a
```

The program must then be relinked:

```
cc .... mymalloc.o -L /usr/local/lib -brtl -bE:myexports
```

In this example, mymalloc.o defines malloc and the export file myexports causes the symbol malloc to be exported from the main program. Calls to malloc from within libc.a will now go to the malloc routine defined in mymalloc.o.

9.6 Developing shared libraries

The way a shared library is used in a development environment is somewhat different to that in a production environment. In a development environment, the library may be constantly changed and altered so that new versions can be tested. On large systems, multiple users may be working with their own version of the shared library. There are a number of things to be aware of to make the development environment for shared libraries easier to use.

If the system has multiple versions of a shared library, then you need to be careful that your program uses the version of the library that you want. This can be achieved with the use of the -L option on the command line and the use of the LIBPATH or LD_LIBRARY_PATH (depending on platform) environment variable.

When an application is started, the system loader reads the loader section of the header of the executable file. It reads the dependency information for any shared objects the executable requires and attempts to load the code for those shared objects into the system shared object segment, if they are not already loaded. Shared objects that are loaded into the system shared library segment have an attribute called the use count. Each time an application program that uses the shared object is started, the use count is incremented. When an application terminates, the use count for any shared objects it was using is decreased by one. When the use count for a shared object in the system shared library segment reaches zero, the shared object is not unloaded, but instead, remains in memory. This is done to reduce the overhead of starting any more applications that use the shared object, because they will not have to load the object into the system shared segment.

9.6.1 The genkld command (Power only)

The `genkld` command is used to list the shared objects that are loaded in the system shared library segment. The output of the command can contain multiple duplicate entries and be quite lengthy, so it is best to filter the output using the `sort` command or by performing a `grep` for the shared object you are investigating. For example:

```
# genkld | sort -u

      d00005c0          19f26f /usr/lib/libc.a/shr.o
      d01a00f8             87a /usr/lib/libcrypt.a/shr.o
      d01a7100            78b4 /usr/lib/libi18n.a/shr.o
      d01af100           137fe /usr/lib/libiconv.a/shr4.o
      d01c3100           124f1 /usr/lib/libodm.a/shr.o
      d01d6100            bc4c /usr/lib/libcfg.a/shr.o
      d01e2880           19583 /usr/lib/libsm.a/shr.o
      d01fc100           262fc /usr/lib/liblvm.a/shr.o
      d02230f8            1624 /usr/lib/libpthreads_compat.a/shr.o
```

The command can only be executed by the root user or a user in the system group. The three columns show the virtual address of the object within the system segment, the size of the object, and the name of the file that was loaded.

9.6.2 The slibclean command

The `slibclean` command can be used by the root user to unload all shared objects with a use count value of zero from the system shared library segment. This command is useful in an environment when shared libraries are under development. You can run the `slibclean` command followed by the `genkld` command to ensure that the shared objects under development are not loaded in the system shared library segment. This means that any application started after this will automatically use the latest version of the shared objects, since the system loader will search for and load them. It also prevents multiple versions of the same objects existing in the system segment.

During the development of shared objects, you may sometimes see an error message similar to the following when creating a new version of an existing shared object:

```
# make libone.so
        cc -O -c source1.c
        cc -berok -G -o libone.so source1.o
ld: 0711-851 SEVERE ERROR: Output file: libone.so
        The file is in use and cannot be overwritten.
make: 1254-004 The error code from the last command is 12.
```

The error message means that the shared object in question has been loaded into the system shared library segment. The file is marked as in use, even if the use count is zero. Running the `slibclean` command will unload all of the unused shared objects from the system. An alternative (and simpler) method of avoiding this problem is to use the `rm -f` command to remove the shared object before creating it.

9.6.3 The dump command

The `dump` command is used to examine the header information of executable files and shared objects. Although both Power and Itanium-based system versions of AIX 5L have a `dump` command, the options and format of the output is completely different. This is because of the difference in executable file formats on the two platforms.

The remainder of this section is applicable to the Power platform only, because the information is used when tracing missing symbols or object from shared libraries. AIX 5L for Itanium-based systems does not include much of this information in the header of the ELF executables, because the system uses run-time linking. See Section 9.6.5, “The ldd and nm commands” on page 288 for information on commands that can be used on Itanium-based systems to investigate shared object problems.

The main options for `dump` that are useful when working with shared libraries are the -H option and the -Tv options.

9.6.3.1 The dump -H command

The `dump -H` command is used to determine which shared objects an executable or shared object depends on for symbol resolution at run time. The interesting information is in the last section of output and has the title ***Import File Strings***. A sample output is as follows:

```
# dump -H example

example:

                        ***Loader Section***
                      Loader Header Information
VERSION#         #SYMtableENT     #RELOCent        LENidSTR
0x00000001       0x00000006       0x0000000e       0x00000047

#IMPfilID        OFFidSTR         LENstrTBL        OFFstrTBL
0x00000003       0x00000158       0x00000019       0x0000019f


                        ***Import File Strings***
INDEX  PATH                          BASE                MEMBER
0      /tmp/addlib/old/complex:/usr/lib:/lib
1                                    libc.a              shr.o
```

```
2                                    libone.a            shr1.o
```

The number of INDEX entries will depend on how many shared objects the target depends on for symbol resolution. The INDEX 0 entry is a colon separated list of directories. If the LIBPATH environment variable is not set when the executable is started, the directories listed in the INDEX 0 entry are searched for by the shared objects mentioned in subsequent entries. The directories in the entry are those used with the -L option when the object was linked. The /usr/lib and /lib entries are always present. If you want these directories to be searched first, you need to add them explicitly to the linker command line and ensure that they appear before any other -L options. Using the example shown above, altering the -L options on the `link` command line to be -L/usr/lib -L/lib -L/tmp/addlib/old/complex would result in an INDEX 0 entry of:

```
0       /usr/lib:/lib:/tmp/addlib/old/complex:/usr/lib:/lib
```

The format of the other entries is as follows:

Index	The index number of the entry in the Import File Strings section.
Path	Optional path name component of the shared object. A path name will be present if a path name was used when the shared object was specified on the `link` command line. The -bnoipath linker option can be used to prevent the path name used on the command line from appearing in this portion of the entry. The -bipath option is the default. The option effects all shared objects listed on the command line.
Base	The name of the archive library containing the shared object, or the name of the shared object itself, if it is a new style shared object.
Member	The name of the shared object if it is contained in an archive library.

Some examples of different `link` commands are appended below, along with the Import File Strings section of the output of the `dump -H` command on the resulting executables. This demonstrates the relationship between the way the shared objects are specified on the command line and the entries in the Import File Strings section of the executable header.

This sample shows the use of the absolute path name on the link line. The following command:

```
cc -o example main.c /tmp/addlib/old/complex/libone.a
```

results in an Import File Strings section of:

```
                        ***Import File Strings***
INDEX  PATH                          BASE                MEMBER
0      /usr/lib:/lib
1                                    libc.a              shr.o
2      /tmp/addlib/old/complex       libone.a            shr1.o
```

This sample shows how to suppress the absolute path name used on the link line. The following command:

```
cc -o example main.c -bnoipath /tmp/addlib/old/complex/libone.a
```

results in an Import File Strings section of:

```
                        ***Import File Strings***
INDEX  PATH                          BASE                MEMBER
0      /usr/lib:/lib
1                                    libc.a              shr.o
2                                    libone.a            shr1.o
```

When using a new style shared object, there is no member entry in the output, only a base entry. The following command:

```
cc -brtl -o example main.c -L/tmp/addlib/new/complex -lone
```

results in an Import File Strings section of:

```
                        ***Import File Strings***
INDEX  PATH                          BASE                MEMBER
0      /tmp/addlib/new/complex:/usr/lib:/lib
1                                    libone.so
2                                    libc.a              shr.o
3                                    librtl.a            shr.o
```

9.6.3.2 The dump -Tv command

The `dump -Tv` command is used to examine the symbol information of a shared object or executable. It lists information on the symbols the object is exporting. It also lists the symbols the object or executable will try and import at load time and, if known, the name of the shared object that contains those symbols. The main columns to examine in the output are headed IMEX, IMPid, and Name.

The IMEX column indicates if the symbol is being imported (IMP) or exported (EXP). The IMPid field contains information on the shared object that the symbol will be imported from. The Name field lists the name of the symbol. For example:

```
# dump -Tv libone.so

libone.so:

                        ***Loader Section***

                        ***Loader Symbol Table Information***
[Index]      Value      Scn     IMEX Sclass   Type           IMPid Name

[0]      0x00000000    undef      IMP     DS EXTref   libc.a(shr.o) printf
[1]      0x00000000    undef      IMP     DS EXTref       libtwo.so function2
[2]      0x20000264    .data      EXP     DS SECdef        [noIMid] function3
[3]      0x20000270    .data      EXP     DS SECdef        [noIMid] function1
```

The output shown above for the libone.so new style shared object indicates that the symbols function1 and function3 are being exported from this object. The object also has two imported symbols on which it depends. The symbol printf is being imported from the shared object shr.o, which is a member of the libc.a archive library. It also imports the symbol function2 from the new style shared object libtwo.so.

9.6.4 Using a private shared object

When used under normal circumstances, a shared object is loaded into the system global shared object segment. Subsequent executables that use the shared object benefit from the fact that it is already loaded.

In a development environment, particularly on a system with multiple developers, it may be preferable to use a private copy of a shared object. This may be useful when developing and testing new functionality in a shared object that is specific to a particular version of the application that a single developer is working on.

If the shared object or container has the access permissions modified (as detailed below), then when the system loader starts an application that uses this shared object, the shared object text will be loaded into the process private segment rather than the system shared object segment. The shared object data will also be loaded into the process private segment instead of its normal location of the process shared object data segment. This means every application will have its own private copy of the shared object text and data. Applications normally have their own copy of the shared object data and share the text with other applications.

To use a private version of the shared object text and data, modify the access permissions as follows:

- If the shared object is contained in an archive library, remove read-other permission from the archive library.

- If the shared object is a new style shared object, for example lib*name*.so, or a standalone shared object, for example, shrobj.o, then remove read-other permission from the shared object.

The effect of this change can be demonstrated using the following sample code.

The file source1.c is used to make a simple shared object. It contains the following code:

```
struct funcdesc {
   int     codeaddr;
   int     TOCentry;
   int     env;
} * shlibfdesc;

void function1(int a)
{
      shlibfdesc = (struct funcdesc *) function1;
      printf("address of function1 is 0x%p\n",shlibfdesc->codeaddr);
      printf("address of shlibfdesc is 0x%p\n",&shlibfdesc);
}
```

The shared object is linked with a small main application, which contains the following code:

```
struct funcdesc {
   int     codeaddr;
   int     TOCentry;
   int     env;
} * mainfdesc;

int main(int argc, char ** argv)
{
        mainfdesc = (struct funcdesc *) main;
        printf("address of main is 0x%p\n",mainfdesc->codeaddr);
        printf("address of mainfdesc struct is 0x%p\n",&mainfdesc);
        function1();
}
```

A function pointer in the C language is implemented as a pointer to a structure that contains three entries. The first entry is a pointer to the address of the code, and the second is a pointer to the table of contents entry for the module containing the function. The third entry is a pointer to environment information and is used by certain other languages, such as Pascal.

The shared object is created with the following commands:

```
cc -c source1.c
ld -G -o libone.so source1.o -bexpall -bnoentry -lc
chmod o-r libone.so
```

Note that read-other permission is removed from the shared object. The main routine is then created with the following command:

```
cc -brtl -o example main.c -lone -L.
```

The output from running the program is as follows:

```
./example
address of main is 0x100002f0
address of mainfdesc struct is 0x200027d4
address of function1 is 0x20000150
address of shlibfdesc is 0x2000127c
```

Note that the address of the code for the main routine is in segment 1 (as expected), and the data structure mainfdesc is in segment 2. Because the shared object had read-other permission removed, it was loaded into segment 2 by the system loader. This can be seen with the address of function1 and shlibfdesc, starting with 0x2.

If read-other permission is restored to the shared object, and the program is invoked again, the result is as follows:

```
chmod o+r libone.so
./example
address of main is 0x100002f0
address of mainfdesc struct is 0x200007d4
address of function1 is 0xd040f150
address of shlibfdesc is 0xf001f27c
```

The address of the main routine and the mainfdesc structure have not changed. The address of function1 now starts with 0xd. This indicates the code is in segment 13, the system shared object segment. The address of the data object shlibfdesc now starts with 0xf, which indicates it is in segment 15, the process private shared object data segment.

9.6.5 The ldd and nm commands

In AIX 5L on Itanium-based systems, the `ldd` and `nm` commands can be used to examine the shared object and symbol dependencies of an executable.

9.6.5.1 The ldd command

When used without flags, the `ldd` command lists the shared objects that will be loaded to start the executable. For example:

```
itsoia64:/home/richard/new>ldd example
example needs:
        libone.so => ./libone.so
        /usr/lib/ia64l32/libc.so.1
```

When used with the -r option, the `ldd` command will attempt symbol resolution, and print error messages if it encounters any problems with unresolved symbols. For example:

```
itsoia64:/home/richard/new>ldd -r example
example needs:
        libone.so => ./libone.so
        /usr/lib/ia64l32/libc.so.1
dynamic linker: example: relocation error: symbol not found: fred;
referenced from: ./libone.so
```

9.6.5.2 The nm command

The `nm` command can be used to list the unresolved symbols in an executable or shared object. Unlike the `dump` command on Power systems, it is not possible to obtain information on the shared object that is expected to supply the symbol. This is because AIX 5L for Itanium-based systems uses run-time linking, and the information on the source of symbols is not kept in the executable. The plus side to this of course is that creating a shared object is much easier in the first place.

9.7 Programatic control of loading shared objects

The dlopen() family of subroutines is supported on the AIX operating system. The functions include:

- dlopen
- dlclose
- dlsym
- dlerror

When used appropriately, they allow a program to dynamically load shared objects into the address space, use functions in the shared object, and then unload the shared object when it is no longer required.

9.7.1 The dlopen subroutine

The dlopen function is used to open a shared object, and dynamically map it into the running programs address space. The specification of the function is as follows:

```
#include <dlfcn.h>

void *dlopen (FilePath, Flags);
const char *FilePath;
int Flags;
```

The FilePath parameter is the full path to a shared object, for example, shrobj.o or libname.so. It can also be a path name to an archive library that includes the required shared object member name in parenthesis, for example, /lib/libc.a(shr1.o).

The Flags parameter specifies how the named shared object should be loaded. The Flags parameter must be set to RTLD_NOW or RTLD_LAZY. If the object is a member of an archive library, the Flags parameter must be ORed with RTLD_MEMBER.

The subroutine returns a handle to the shared library that gets loaded. This handle is then used, with the dlsym subroutine, to reference the symbols in the shared object. On failure, the subroutine returns NULL. If this is the case, the dlerror subroutine can be used to print an error message.

9.7.2 The dlsym subroutine

The dlopen subroutine is used to load the library. If successful, it returns a handle for use with the dlsym routine to search for symbols in the loaded shared object. Once the handle is available, the symbols (including functions and variables) in the shared object can be found easily. For example:

```
lib_func=dlsym(lib_handle, "locatefn");
error=dlerror();
if (error)
{
   fprintf(stderr, "Error:%s \n",error);
   exit(1);
}
```

The dlsym subroutine accepts two parameters. The first is the handle to the shared object returned from the dlopen subroutine. The other is a string representing the symbol to be searched for.

If successful, the dlsym subroutine returns a pointer that holds the address of the symbol that is referenced. On failure, the dlsym subroutine returns NULL. This, again, can be used with the dlerror subroutine to print an error message.

9.7.3 The dlclose subroutine

The dlclose subroutine is used to remove access to a shared object that was loaded into the processes' address space with the dlopen subroutine. The subroutine takes, as its argument, the handle returned by dlopen.

9.7.4 The dlerror subroutine

The dlerror subroutine is used to obtain information about the last error that occurred in a dynamic loading routine (that is, dlopen, dlsym, or dlclose). The returned value is a pointer to a null-terminated string without a final new line. Once a call is made to this subroutine, subsequent calls without any intervening dynamic loading errors will return NULL.

Applications can avoid calling the dlerror subroutine in many cases by examining errno after a failed call to a dynamic loading routine. If errno is ENOEXEC, the dlerror subroutine will return additional information. In all other cases, dlerror will return the string corresponding to the value of errno.

Note

The dlerror() subroutine is not thread-safe, because the string may reside in a static area that is overwritten when an error occurs.

9.7.5 Using dynamic loading subroutines

In order to use the dynamic loading subroutines, an application must be linked with the libdl.a library. The shared objects used with the dynamic loading subroutines can be traditional AIX shared objects, or shared objects that have been enabled for run-time linking with the -G linker option.

When the dlopen subroutine is used to open a shared object, any initialization routines specified with the -binitfini option, as described in Section 9.2.4, "Initialization and termination routines" on page 270, will be called before dlopen returns. Similarly, any termination routines will be called by the dlclose subroutine.

9.7.6 Advantages of dynamic loading

The use of dynamic linking allows several benefits for application developers:

1. The ability to share commonly used code across many applications, leading to disk and memory savings.
2. It allows the implementation of services to be hidden from applications.

3. It allows the re-implementation of services, for example, to permit bug and performance fixes or to allow multiple implementations selectable at run time.

9.8 Shared objects and C++

The C++ language, although similar in some respects to the C language, offers many additional facilities. One of these is known as *function overloading*, which makes it possible to have multiple functions with the same name but different parameter lists. This feature means it is not possible to use the function name alone as a unique identifier in the symbol table of an object file. For this reason, function names in C++ are *mangled* to produce the symbol name. The mangling uses a code to indicate the number, type, and ordering of parameters to the function.

It is the name mangling feature of C++ that means the process of creating a shared object, that includes object code created by the C++ compiler, is slightly more complicated than when using code produced by the C compiler.

Although it would be possible to manually create import and export files, the process is time consuming, because a unique symbol name is required for each instance of an overloaded function.

9.8.1 Generating an exports file on Power

A very useful option to the `makeC++SharedLib` command is the ability to save the export file that is generated behind the scenes and normally discarded after use. If saved, this export file can then be used as an import file when creating another shared object. The -e expfile option is used to save the export file. Note that the export file produced does not have an object file name field (#!) on the first line, so one will have to be manually added, if required. The makeC++SharedLib shell script is supplied with the compiler on Power platforms.

9.8.2 The -qmkshrobj option

The -qmkshrobj option is used to instruct the C++ compiler to create a shared object from previously created object files and archive libraries. This option also makes it much easier to create shared objects that use template functions. Refer to Section 11.5, “Shared objects with templates” on page 406 for more information.

For example, to create a shared object shr1.o from the files source1.o and source2.o, use the following command:

```
xlC -qmkshrobj -o shr1.o source1.o source2.o
```

The -G option can also be used in conjunction with the -qmkshrobj option to create an object that uses the new style naming convention and is enabled for run-time linking. This is the method that should always be used on AIX 5L for Itanium-based systems. For example:

```
xlC -G -qmkshrobj -o libshr1.so source1.o source2.o
```

To specify the priority of the shared object, which determines the initialization order of the shared objects used in an application, append the priority number to the -qmkshrobj option. For example, to create the shared object shr1.o, which has an initialization priority of -100, use the following command:

```
xlC -qmkshrobj=-100 -o shr1.o source1.o source2.o
```

On the Power platform, if none of the -bexpall, -bE:, -bexport:, or -bnoexpall options are specified, then using the -qmkshrobj option will force the compiler to generate an exports file that exports all symbols. This file can be saved for use as an import file when creating other shared objects, if desired. This is done using the -qexpfile=filename option. For example:

```
xlC -qmkshrobj -qexpfile=shr1.exp -o shr1.o source1.o source2.o
```

9.8.3 Mixing C and C++ object files

In addition to the mangling of symbol names, the C++ language may differ from the C language in the way function arguments are passed on the calling stack. The C++ language allows an extern definition to specify a linkage convention. We can speak of having a function having C or C++ linkage.

When mixing C and C++ code together, it is necessary to use a linkage block to specify a C routine that will be called from a C++ routine. This prevents the compiler from mangling the name of the C routine, which would result in a symbol name that could not be resolved. For example, to call the C function foo from C++ code, the declaration of foo must be in an external linkage block:

```
extern “C” {
void foo(void);
}
class1::class1(int a)
{
   foo();
}
```

If the declaration of foo was not contained in the extern “C” block, the C++ compiler would mangle the symbol name to foo__Fv.

When mixing C and C++ objects within a single shared object, either the `makeC++SharedLib` command (which uses the C++ compiler) or the -qmkshrobj option of the C++ compiler should be used to create the shared object. Do not use the C compiler or the linker, because they may not produce the correct result, as they are not aware of C++ constructors, destructors, templates, and other C++ language features.

9.9 Order of initialization

There are situations where the order of initialization of data objects within a program is important to the correct operation of the application. A priority can be assigned to an individual object file when it is compiled. This is done using the -qpriority option. For example:

```
xlC -c zoo.C -qpriority=-50
```

The C++ compiler also supports options that can be used to indicate the relative order of initialization of shared objects. When using the C++ compiler, the priority is specified as an additional value with the -qmkshrobj option. For example:

```
xlC -qmkshrobj=-100 -o shr1.o source1.o
```

Priority values can also be indicated within C++ code by using the priority compiler directive as follows:

```
#pragma priority(value)
```

These values alter the order of initialization of data objects within the object module.

9.9.1 Priority values

Priority values may be any number from -214782623 to 214783647. A priority value of -214782623 is the highest priority. Data objects with this priority are initialized first. A priority value of 214783647 is the lowest priority. Data objects with this priority are initialized last. Priority values from -214783648 to -214782624 are reserved for system use. If no priority is specified, the default priority of 0 is used.

The explanation of priority values uses the example data objects and files shown in Figure 61.

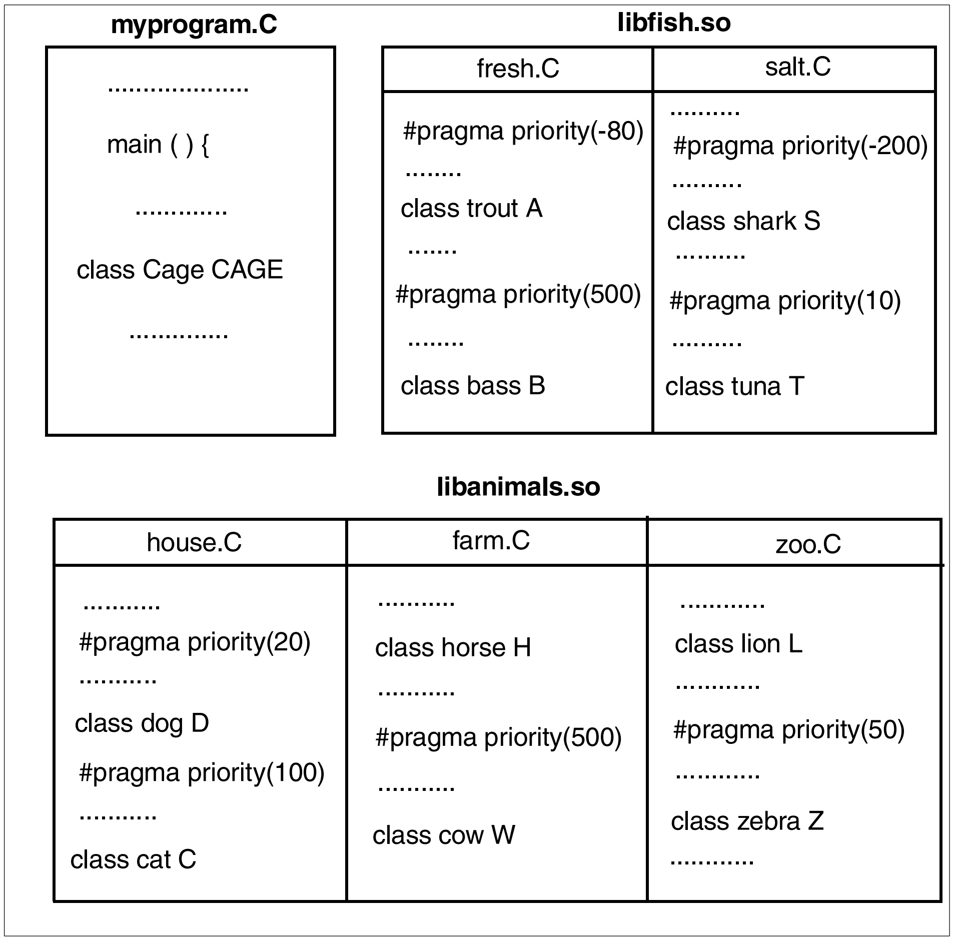


Figure 61. Illustration of objects in fish.o and animals.o

This example shows how to specify priorities when creating shared objects to guarantee the order of initialization. The user should first of all determine the order in which they want the objects to be initialized, both within each file and between shared objects:

1. Develop an initialization order for the objects in house.C, farm.C, and zoo.C:

   a. To ensure that the object lion L in zoo.C is initialized before any other objects in either of the other two files in the shared object animals.o, compile zoo.C using a -qpriority=nn option, with nn less than zero, so that data objects have a priority number less than any other objects in farm.C and house.C:

```
xlC zoo.C -c -qpriority=-50
```

b. Compile the house.C and farm.C files without specifying the -qpriority=nn option. This means the priority will default to zero. This means data objects within the files retain the priority numbers specified by their `#pragma priority(`nn`)` directives:

```
xlC -c house.C farm.C
```

c. Combine these three files into a shared library. Use the C++ compiler to construct the shared object libanimals.so with a priority of 40:

```
xlC -G -qmkshrobj=40 -o libanimals.so house.o farm.o zoo.o
```

2. Develop an initialization order for the objects in fresh.C and salt.C, and use the `#pragma priority(`value`)` directive to implement it:

   a. Compile the fresh.C and salt.C files

   ```
   xlC -c fresh.C salt.C
   ```

   b. To assure that all the objects in fresh.C and salt.C are initialized before any other objects, including those in other shared objects and the main application, use the C++ compiler to construct a shared object fish.o with a priority of -100:

   ```
   xlC -G -qmkshrobj=-100 -o libfish.so fresh.o salt.o
   ```

   Because the shared object libfish.so has a lower priority number (-100) than libanimals.so (40), when the files are use together by an executable, the objects in libfish.so are initialized first.

3. Compile the main program, myprogram.C, which contains the function main, to produce an object file, myprogram.o. By not specifying a priority, this file is compiled with a default priority of zero, and the objects in main have a priority of zero:

   ```
   xlC -c myprogram.C
   ```

4. Produce an executable file, animal_time, so that the objects are initialized in the required order. Enter:

   ```
   xlC -o animal_time main.o -L. -lfish -lanimals
   ```

When the animal_time executable is run, the order of initialization of objects is as shown in Table 82.

Table 82. Order of initialization of objects in prriolib.a

Object	**Priority value**	**Comment**
libfish.so	-100	All objects in fish.o are initialized first because they are in a library prepared with `xlC -qmkshrobj=-100` (lowest priority number, -100 specified for any files in this compilation).
shark S	-100(-200)	Initialized first in fish.o because within file, `#pragma priority(-200)`.
trout A	-100(-80)	`#pragma priority(-80)`.
tuna T	-100(10)	`#pragma priority(10)`.
bass B	-100(500)	`#pragma priority(500)`.
myprog.o	0	File generated with no priority specifications; default is 0.
cage CAGE	0(0)	Object generated in main with no priority specifications; default is 0.
libanimals.so	40	File generated with `xlC -qmkshrobj=40`.
lion L	40(-50)	Initialized first in file animals.o compiled with -qpriority=-50.
horse H	40(0)	Follows with priority of 0 (since -qpriority=nn is not specified at compilation and no `#pragma priority(`nn`)` directive is given).
dog D	40(20)	Next priority number (specified by `#pragma priority(20)`).
zebra N	40(50)	Next priority number from `#pragma priority(50)`.
cat C	40(100)	Next priority number from `#pragma priority(100)`.
cow W	40(500)	Next priority number from `#pragma priority(500)`.

9.10 Troubleshooting

The following tips and hints can be used to help link and load of C and C++ programs on AIX 5L.

9.10.1 Link failures on Power

When linking large applications with many libraries, the linker may exit with some strange errors referring to BUMP or indicating that the binder was killed. This may be because of low paging space or because of low resource limits for the user invoking the command. The AIX linker offers a great deal more functionality than traditional UNIX linkers, but it does require a reasonable amount of virtual memory, particularly when linking large applications with many libraries.

If this type of error is encountered, check the paging space available on the machine. In addition, check the resource limits for the user invoking the linker. This can be done with the `ulimit` command.

9.10.1.1 Unresolved symbols

When linking your application with many libraries, particularly those supplied by a third party product, such as a database, it is not unusual during the development cycle to see a linker error warning of unresolved symbols.

The linker supports options that can be used to generate linker log files. These log files can then be analyzed to determine the library or object file that references the unresolved symbol. This can help in tracking interdependent or redundant libraries being used in error.

The -bmap:filename option is used to generate an address map. Unresolved symbols are listed at the top of the file, followed by imported symbols.

The -bloadmap:filename option is used to generate the linker log file. It includes information on all of the arguments passed to the linker along with the shared objects being read and the number of symbols being imported. If an unresolved symbol is found, the log file produced by the -bloadmap option lists the object file or shared object that references the symbol. In the case of using libraries supplied by a third party product, you can then search the other libraries supplied by the product in an effort to determine which one defines the unresolved symbol. This is particularly useful when dealing with database products that supply many tens of libraries for use in application development.

9.10.2 Run time tips

If large parts of the shared libraries are paged in all at once because of C++ calls or many references between libraries, it may be faster to read the library rather than demand-page it into memory. Remove read-other permission from the applications shared libraries and see if the loading performance improves. If it does, then reset the original permissions and set the following environment variable:

```
LDR_CNTRL = PREREAD_SHLIB
```

By using this environment variable, the libraries are read very quickly into the shared memory segment.

9.11 Linker differences on Itanium-based systems

This section describes a list of the major differences between the linker on AIX 5L for Power and the linker on AIX 5L for Itanium-based systems.

9.11.1 libelf.so instead of libld.a

Since the module format on AIX 5L for Itanium-based systems is ELF, the XCOFF specific processing library libld.a is not available. Instead, libelf.so provides the object file specific processing code for the ELF environment.

9.11.2 Mixed mode linking no longer valid

On AIX 5L for Power, an archive may contain different types of objects. For example, libc.a contains .o files, 32-bit shared libraries, and 64-bit shared libraries. Applications (both 32-bit and 64-bit) can link dynamically and/or statically against this single archive (libc.a). This behavior is not duplicated on AIX 5L for Itanium-based systems. The System V.4 semantics for shared library creation (static and dynamic linking), does not allow for a single archive to be used in the same manner as on AIX 5L for Power. For AIX 5L for Itanium-based systems, library variants exist in separate directories as follows:

```
/usr/lib/ia64l32 (ILP32 little-endian libraries)
/usr/lib/ia64l64 (LP64 little-endian libraries)
```

Mixed mode linking is disallowed. The linker's implementation determines its target mode (ia64l32 or ia64l64). When a directory specifier is provided, such as 'ld ... -Ldir ...', the linker will add that directory to its search list before the default directories to be searched (as it always has). When the run-time linker and link-editor are searching for libraries, it is not a fatal error if an ELF file of the wrong type is encountered in the search. Instead, the link-editors will exhaust the search of all paths before determining a matching object could not be found. This will permit having a common search path (LD_LIBRARY_PATH) that contains a mix of directories containing differing process models.

9.11.3 Symbol resolution performed by run-time linker

On AIX 5L for Itanium-based systems, symbol resolution in a running program is performed by the run-time linker (in libc), according to the rules specified in the generic IA-64 ABI. At link time, shared objects referenced on the command line are listed in the "needed module" section of the program. Symbols are resolved at run time by sequentially searching the needed modules list until a definition is found. On AIX 5L for Power, symbol resolution is performed at link time. If a symbol is defined in a shared object (or module referenced on the command line, the linker records the symbol name and defining module in the loader section of the program. When a program is executed, imported symbols must be resolved by finding them in the defining module, as recorded at link time. On both platforms, an import file can be used in place of a shared object (module) that is not available or has not yet been built. The inability to associate a symbol to a specific library on AIX 5L for Itanium-based systems makes the order in which the libraries are specified very important. For example, with respect to rpc, if you want the streams behavior, your library order must be `libnsl`, then `libc`, and if you want sockets behavior, it must be `libc` followed by `libnsl`. For example:

1. Module A contains a definition for function X
2. Library foo also has a definition for a different function X
3. Library foo has a function Y, which calls function X

On AIX for Power, function Y will get the X present in library foo. On AIX 5L for Itanium-based systems, function Y will get the X present in Module A.

9.11.4 AIX system calls for binding

The AIX load(), loadquery() and unload() APIs continue to be supported on AIX 5L for Itanium-based systems, however, loadbind() is not supported.

9.11.5 Linker options

The AIX 5L for Power linker differs from the SVR4 style linker used in AIX 5L for Itanium-based systems in features and functionality. The flags used, as well as their default behavior, are often different. The AIX 5L for Itanium-based systems linker does not support relinking of an executable and will ignore the following AIX flags because they either have no meaning in an ELF and IA-64 environment or they have no equivalent:

```
-k -z -basis -bautoexp -bbigtoc -bbindcmds:File -bbinder:File
-bbindopts:File -bcomprld -bcrld -bcror15 -bcror31 -bD:Number -bdbg:Option
-bdebugopt:Option -bdelcsect -berrmsg -bex1:File -bex2:File -bex3:File
-bex4:File -bex5:File -bgc -bgcbypass:Number -bglink:File -bh:Number
-bhalt:Number -bI:File -nimport:File -binitfini:Init:Term:Priority -bipath
-bkeepfile:File -blazy -bL:File -bloadmap:File -bM:ModuleType
-bmodtype:ModuleType -bmaxdata:Number -bmaxstack:Number -bS:Number -bnl
-bnoloadmap -bnoautoimp -bnobigtoc -bnobind -bnocomprld -bnocrld
-bnodelcsect -bnoentry -bnoerrmsg -bnogc -bnoglink -bnoipath -bnolibpath
-bnom -bnoobjreorder -bnop:Nop -bnoquite -bnoreorder -bnortl -bnortllib
-bnostrip -bnosymbolic -bnotextro -bnro -bnotypchk -bnov -bnox -bquiet -br
-breorder -brename:Old -brtl -brtllib -bS:number -bsmap -bstabcmpct
-bsxref:File -btypchk -bx -bX:File -bxref:File -m -M -SNumber -uName -v
-zString
```

There are other AIX linker flags for which there are corresponding AIX 5L for Itanium-based system linker flags that provide either identical or similar functionality. Table 83 details the AIX linker flags and their semantics with the corresponding AIX 5L for Itanium-based systems linker flags and semantics.

Table 83. Linker flag comparison

AIX 5L for Power linker flag	AIX 5L for Itanium-based systems equivalent flag
-DNumber locates the initialized data.	Possible with the Map File (-M option).
-eLabel sets the entry point to Label.	-e epsym sets the entry point to epsym.
-G produces a shared object.	-G produces a shared object.
-HNumber aligns the text, data, and loader sections at a multiple of number.	Possible with the Map File (-M option).
-lName processes libName.a in static mode.	-l x
-LDir adds Dir to the list of search directories.	-L path
All -L options are processed first, before any of the -l options; therefore, each library specified with the -l option will be searched in all directories specified.	Each directory specified with the -l option is searched in the directory specified with -L only if -L precedes the -l option.

AIX 5L for Power linker flag	AIX 5L for Itanium-based systems equivalent flag
LDPATH = dirlist Libraries are searched in the following order: 1. Directories specified by -L or LIBPATH if there is no -L option. 2. Standard directories (/usr/lib and /lib).	LD_LIBRARY_PATH = dir1:dir2; dir3:dir4 Libraries are searched in the following order: 1. dir1 and dir2. 2. directories specified by the -L option. 3. dir3 and dir4. 4. standard directories (/usr/lib and /usr/ccs/lib).
-oName names the output file.	-o outfile names the output file.
-r produces non-executable output file suitable for another linking.	-r produces a relocatable file (partially linked file).
-s strips symbolic debugging information.	-s strips symbolic debugging information.
-TNumber sets the starting address of the text section. It does not have any effect on run-time addresses.	Possible with the Map File (-M option).
-bautoimp or -bso imports symbols from any shared object specified as input. This option is valid for all shared objects.	-Bdynamic links with the shared object version of a library (when available). This option is valid only for the shared objects following it and until the next -Bstatic.
-bC:File or -bcalls:File writes an address map to a file.	-m produces a memory map of input/output sections.
-bdynamic or -bshared processes subsequent shared objects in dynamic mode.	-Bdynamic links with the shared object version of a library (when available) until the next -Bstatic.
-bstatic processes subsequent shared objects in static mode.	-Bstatic links with the archive version of a library until next -Bdynamic.
-bE:File or -bexport:File exports the external symbols listed in File.	-Bexport:File exports all global and weak symbols listed in File.
-bernotok or -bf reports an error if there are any unresolved external references.	-z defs do not allow undefined symbols (default for executables).
-berok allows unresolved external references in the output file.	-z nodefs allows undefined symbols (default for shared objects).
-bexpall exports all global symbols.	-Bexport.

AIX 5L for Power linker flag	AIX 5L for Itanium-based systems equivalent flag
-blibpath:Path uses Path when writing the loader section of the output file.	-R path records path for run-time library search (this may not be equivalent).
-bnoexpall does not export any symbol not listed in the export file (default).	-Bexport:File hides all symbols except those listed in File.
-bpD:Origin specifies origin as the address of the first byte of the file page containing the beginning of the data section.	Possible with the Map File (-M option).
-bpT:Origin specifies origin as the address of the first byte of the file page containing the beginning of the text section.	Possible with the Map File (-M option).
-bro or -btextro ensures that there are no load-time relocations for the text section.	-z text in dynamic mode only. Do not allow relocations against non-writable allocatable segment.
-bsymbolic assigns the symbolic attribute to most symbols exported without an explicit attribute.	-Bsymbolic=[list I :File] binds all references to the named symbol to its definition.
LIBPATH environment variable.	-YP, dirlist changes default library search directories.
-bnso -bI:/lib/syscalss.exp.	-a produces a statically-linked executable file (must not be used with -r and -dy options).

Refer to the `ld` product documentation for a full description of all of the linker options.

9.11.6 Import/export file support

As an accommodation for existing AIX applications, the AIX 5L for Itanium-based systems linker provides support for the use of Import and Export files. This support is different from the support on AIX 5L for Power in both syntax and semantics which may require makefile changes. The command line arguments to `ld` are:

```
-Bimport:file
-Bexport:file
```

The difference is that the option is specified with -B instead of -b. In addition, the optional short form (equivalent to -bI:file and -bE:file) is not supported.

9.11.6.1 Import/export file syntax

Directives begin with a %, which must be in column one. There is currently one directive, %soname. The directive is followed by a string (quoted or not):

```
%soname <name>
```

This directive has the effect of adding the named shared object to the needed list of the object being built. (The needed objects of a dynamic executable or shared object can be listed with the -Lv option of the `dump` command.) The comment character is #. It may occur in any column. The comment goes to the end of line. Symbols should be listed one per line. Symbol entries may contain an optional tag which may have an optional number:

```
sym_name [{ws} tag [{ws} decimal_number]] {eol}
```

where tag is one of syscall, function, object, and decimal_number is a system call number for the syscall tag and a size for an object. {ws} stands for white space, either space or tab; {eol} stands for end-of-line. In an export file, symbols with the syscall tag will be marked for export by the system loader as system calls. All symbols not marked for export will not be visible outside of the shared object defining them. The %soname directive is ignored in an export file. Multiple import files can be specified on the `ld` command line, but only one export file.

The -Bsyscall option is still present and functioning, but its functionality is now additionally obtainable through export files. The -Bsyscall option will be removed in the future.

9.11.7 Shared library

The creation process of shared library on AIX 5L for Itanium-based systems is somewhat different from that of AIX for Power.

9.11.7.1 Creating a shared object

The steps to create a shared object are as follows:

1. Compile object files:

   ```
   $ cc -c mine. c common.c
   ```

2. Create export file if required. The linker exports all global symbols of a shared object. You can use an export file to export only symbols needed to be visible to the outside of the object and hide the rest. For example, we want to export only a function named myentry and a global variable named visit. We create an export file export.mine and put those two symbols in it:

   ```
   %soname /home/guest/libmine.so
   ```

```
   myentry
   visit
```

3. Link with the -G option:

```
   $ ld -G -o libmine.so mine.o common.o -Bexport:export.mine -lc
```

You can set the name that will be recorded by other objects when they are linked against this object, and which will subsequently be used to locate the library at run time if it contains a slash (`/`) (the SONAME dynamic section entry) with the -h option:

```
$ ld -G -o libmine.so mine.o common.o -h/home/guest/libmine.so
-Bexport:export.mine -lc
```

9.11.7.2 Examine a shared library

You can examine the dynamic section information with the `dump` command:

```
$ dump -Lv libmine. so
libmine. so:
**** DYNAMIC SECTION INFORMATION ****
[INDEX] Tag Value
[1] SONAME /home/guest/libmine.so
[2] ...
```

9.11.7.3 Link a shared library

Once the shared object has been installed into the desired location, in this case, /home/guest, you can then link an executable against this object as follows:

```
$ cc -o main main. c -Bimport:import.mine -L/home/guest -lmine
```

You need to import the global symbol from an import file. In the case of libmine.so, the content of the import file can be exactly like export file, except that the %soname is ignored in the export file by the static linker `ld`.

Chapter 10. POSIX threads

This chapter covers porting issues with respect to the IEEE POSIX 1003.1 1996 thread interface standard (or just POSIX threads). This standard can be found at:

```
http://standards.ieee.org
```

For certain downloads and access to information from this site, you are required to register as a member.

Although this standard dates back to 1996, there are running applications that are based on thread libraries not conforming to this standard.

Furthermore, the POSIX threads standard consists of both requirements and options. Certain aspects are left implementation defined, unspecified, or even undefined. Hence, running applications based on the same source code on platforms having POSIX threads conforming implementations may result in a different behavior.

In an attempt to best support the reader's porting task, this section gives an introduction to POSIX threads (to support the case where the system being ported from does not support POSIX threads), provides coding examples, lists porting issues, summarizes recommendations at the end of each subsection, and ends with a quick reference to the thread library.

AIX 5L conforms to the POSIX threads specification.

10.1 Introduction to threads

A thread is a part of a process that can execute a set of instructions independently. All threads of a process share its data, but they can also create an area for private data.

The scheduling of threads resembles that of processes, but there is no inherent parent/child hierarchy between threads. They are all equal within a process and identified by a thread ID.

Synchronization of threads' access to shared data can be governed by the use of mutual locks (mutexes) and condition variables.

Threads are often more efficient than processes. They start up faster, require less overhead, simplify access to shared data, and are easier to synchronize.

The following section gives an introduction to POSIX threads.

10.1.1 Threads versus processes

A characterizing feature of UNIX-based operating systems is their ability to multi-task, that is, execute processes simultaneously. The individual processes' execution flows are either interleaved and/or non-interleaved, depending on the underlying hardware.

10.1.1.1 Processes

Each process has its own run-time environment. It includes, among other things, a set of identifiers (for example, the PID and GID), a current directory, an environment (variables and their associated values), a set of file descriptors, a file-mode creation mask, a signal mask, a set of pending signals, and a private address space (including a stack and a heap).

New processes are created by forking a child process. Child processes are themselves processes, so they have their own environment. When created, the environment of a child process inherits/copies certain values from its parent process. Values are being passed implicitly from one process to another, that is, from one address space to another. Except for shared memory, changes in one process' address space does not cause a change in another process' address space.

Once a process has been forked, any communication between the parent and child process, or for that matter two processes not directly related by a parent-child relationship, must be done explicitly. This can be achieved using interprocess communication (IPC).

For this purpose, UNIX-based OSes support constructs such as named and unamed pipes, shared memory, sockets, and semaphores. Developing applications using these constructs can be an intellectually challenging task. The challenge can usually be traced back to the fact that resources may have to be shared among more than one process.

To avoid inconsistency of shared resources, processes sometimes need exclusive access to a resource. A process may need to perform several computational steps in order to complete a logically atomic step on a shared resource. Semaphores provide a way of preventing other processes from disturbing a single process' logical step (for example, updating some shared memory), even while it has been preempted.

While semaphores provide a possible solution to this challenge, they may also introduce unwanted situations, such as deadlocks, if not used carefully. Because the fork routine also copies semaphores and their state, a child process may grab a semaphore and terminate without having released a semaphore. The use of sockets and pipes for IPC can also be challenging when developing code.

When a child process terminates, it can return an integer value to its parent. If more information needs to be returned, IPC may be needed or the child process may even use a file for the purpose.

Although exchanging data correctly between separate processes can seem demanding from a coding point of view, there are many reasons why one would want processes to communicate.

For example, on a multiprocessor system, one may reduce the overall time required to solve a computational problem by executing processes in parallel on several processors. But at the same time, the correctness of the computation may depend on the processes exchanging data, that is performing IPC.

Another reason for having several processes executing concurrently may simply be that it is simpler, both conceptually and from a design point of view, if one masters IPC.

However, from the operating systems point of view, forking processes and handling IPC can be computationally “expensive.” Forking processes takes time and can require considerable amounts of memory.

10.1.1.2 Threads

Threads provide a “cheaper” alternative, when there is no explicit need for creating separate processes, but merely a need to have some degree of parallel execution and manipulation of the same global memory.

Threads are created within a process. They are schedulable entities, which share the process’:

- Program instructions
- Global data (address space)
- File descriptors
- Signal handlers
- Environment (variable) and working directory
- Set of identifiers (PID, GID, and so forth)

but have their own flow control and identity:

- Stack

- Thread ID
- Signal mask
- Priority
- Errno
- Program counter/stack pointer and other registers

Access to the process' global data makes exchanging data easy (as opposed to processes which cannot see each others data, except what is placed in shared memory). But there is still a need for mechanisms such as mutual exclusion.

Creating threads is much faster than forking processes and does not require a copy of the forking process's address space. In fact, on systems supporting threading, a process initially has a single implicit thread, which controls the program's main flow.

In this chapter, we shall only be concerned with systems supporting threading.

10.1.1.3 Example of a POSIX threaded program

This section show a simple example of a threaded program. The program prints two words to the standard output. A complete listing of the POSIX thread functions is given in Section 10.1.2.3, "Summary of POSIX threads library routines" on page 313.

```
#include <pthread.h> /* include file for pthreads - include 1st */
#include <stdio.h>

void *my_thread(void *string)
{
 printf("%s\n", (char *)string);
 pthread_exit(NULL);
}

int main(int argc, char **argv)
{
 char *hello = "hello ";
 char *world = "world";

 pthread_t my_th1, my_th2;

 int rc;

 rc = pthread_create(&my_th1, NULL, my_thread, hello);
20  rc = pthread_create(&my_th2, NULL, my_thread, world);
21
22  rc = pthread_join(my_th1, NULL);
23  rc = pthread_join(my_th2, NULL);
24
25  exit(0);
26 }
```

To compile the code, use one of the `_r`-suffixed invocations of the base compilers (for further details, see Section 10.7, "Compiling and linking" on page 362):

```
xlc_r -o hello hello.c
```

Executing the application hello prints the words `hello` and `world` on the screen, not necessarily in that order. In Section 10.2, "Thread scheduling" on page 322, the scheduling of threads is explained. The scheduling controls how threads are run on the underlying operating system and hardware.

When the function main is being run, only one thread exists. In lines 19 and 20, two additional threads are started. For the sake of compactness of the code, we have chosen not to check the return code. This could have been done by the following code:

```
rc = pthread_create(&my_th1, NULL, my_thread, hello)
if (rc == .....) {
   /* process error */
}
```

The calls in line 19 shows that a thread within a process has an associated ID (first argument), a function which it will run (third argument), and a pointer to an argument passed to that function (fourth argument). The second argument controls the thread's attribute. Here, NULL just means that the default values are being used.

Notice that the starting point of flow of control of a created thread is not the same as that of the creating thread. For the fork call, both parent and child process continue from the same point, which is the return from the fork call.

Once the two new threads have been started, the main thread waits for them (line 22-23) to complete (line 7).

Table 84 shows the similarities between some basic operations on processes and threads.

Table 84. Operations similarities for processes and threads

Operation	Processes	Threads
Creation	fork/exec	pthread_create
Termination	exit	pthread_exit
Identification	getpid	pthread_self
Synchronization	waitpid	pthread_join
Yield processor	yield	sched_yield
Sending signals	kill	pthread_kill
Set signal mask	sigsetmask	pthread_sigmask

A larger example of a threaded program is given in Section 10.11, “Example: The Mandelbrot set” on page 391.

10.1.2 Thread library versions

Prior to the POSIX 1003.1 1996 thread interface standard, several draft versions were used as a basis for implementations.

10.1.2.1 Draft versions

There are several draft versions of the POSIX threads standard. Table 85 shows the AIX POSIX thread conformance. The third column specifies how threads (user threads) are mapped to kernel threads. Thread models are discussed in Section 10.2, “Thread scheduling” on page 322.

Table 85. AIX POSIX thread conformance

AIX version and release	Threads version	Thread model
3.2	POSIX Draft 4	M:1
4.1 and 4.2	POSIX Draft 7	1:1
4.3 and 5.1	UNIX 98, POSIX 1003.1 1996, POSIX Drafts 7 and 4	M:N

Other common versions of thread packages are DCE threads, based on The Open Group's DCE 1.2.1 version, which in turn is based on POSIX Draft4. DCE threads are also sometimes referred to as CMA threads.

Implementations of draft versions include routines that are either not named in the final standard or which return values different from the final standard.

AIX 5L provides program support for both Draft 7 of the POSIX Thread Standard and Xopen Version 5 Standard, which includes the final IEEE POSIX 1003.1 1996 thread interface standard.

The libpthreads_compat.a library is provided for program development.

10.1.2.2 Final version

The final version of POSIX threads leaves certain aspects implementation defined, unspecified, or even undefined. This means that porting from another POSIX threads conforming platform to AIX 5L cannot be assumed to not require some amount of work. Getting the code to compile should be relatively straightforward, but issues, such as performance, could turn out to require work. For instance, under AIX 5L, the default ratio between user threads and kernel threads of 8:1 is used. It might be necessary to choose another ratio. This will be discussed in Section 10.2, “Thread scheduling” on page 322.

Using POSIX threads will result in a more portable application code.

10.1.2.3 Summary of POSIX threads library routines

This section classifies and summarizes the POSIX threads library routines in AIX 5L.

The library routines can be classified into four functional categories. There are other ways of classifying the routines; here we merely suggest one. The categories are:

- Thread management
- Execution scheduling
- Synchronization
- Thread-specific data

In Table 86 to Table 91, the functional categories are described by routine names and short descriptions. In Section 10.10, “Quick reference” on page 371, detailed information about the function headers can be found. Rather than setting the errno variable, most routines return an integer value, to be interpreted as an error code. A value of zero indicates that the call was successful. Other values are passed or retrieved through appropriately typed arguments.

Note

To use the thread routines, `#include <pthread.h>` must be included as the *first* header file of each source file using the threads library, as shown in the example in Section 10.1.1.3, “Example of a POSIX threaded program” on page 310. This is because it defines some important macros that affect other system header files. Having the pthread.h header file as the first included file ensures the usage of thread-safe subroutines.

Table 86. Thread management

Name	**Description**
pthread_attr_init	Initializes a thread attributes object.
pthread_attr_destroy	Destroys a thread attributes object.
pthread_attr_setdetachstate	Sets the detachstate attribute of a thread attributes object. This attribute determines if a thread created with this thread attributes object is in a detached state or not.
pthread_attr_getdetachstate	Gets the detach state attribute from a thread attributes object.
pthread_attr_setstackaddr	Sets the value of the stackaddr attribute of a thread attributes object. This attribute specifies the stack address of a thread created with this attributes object.
pthread_attr_getstackaddr	Gets the stackaddr attribute from a thread attributes object.
pthread_attr_setstacksize	Sets the value of the stacksize attribute of a thread attributes object. This attribute specifies the minimum stack size, in bytes, of a thread created with this thread attributes object.
pthread_attr_getstacksize	Gets the stacksize attribute from a thread attributes object.
pthread_testcancel	Creates a cancellation point in the calling thread.
pthread_setcancelstate	Atomically sets the calling thread's cancelability state to the indicated state and returns the previous cancelability state at a specified location reference.
pthread_setcanceltype	Atomically sets the calling thread's cancelability type to the indicated type and returns the previous cancelability type at a specified location reference.

Name	Description
pthread_create	Creates a new thread and initializes its attributes using the thread attributes object specified, or standard values instead, if the NULL pointer is specified. After thread creation, a thread attributes object can be reused to create another thread, or deleted.
pthread_exit	Terminates the calling thread safely, and stores a termination status for any thread that may join the calling thread.
pthread_cancel	Requests the cancellation of the specified thread. The action depends on the cancelability of the target thread.
pthread_kill	Sends the specified signal to the specified thread. It acts with threads like the kill subroutine with single-threaded processes.
pthread_join	Blocks the calling thread until the specified thread terminates. If the specified thread is in a detached state (non-joinable), an error is returned.
pthread_detach	Used to indicate to the implementation that storage for the specified thread can be reclaimed when that thread terminates.
pthread_once	Executes the specified routine exactly once in a process. The first call to this subroutine by any thread in the process executes the given routine, without parameters. Any subsequent call will have no effect.
pthread_self	Returns the calling thread's ID.
pthread_equal	Compares the two specified thread IDs. Returns zero if and only if the IDs are equal.
pthread_atfork	Threads can fork processes. This routine registers fork cleanup handlers. Three handlers can be specified: prepare, parent, and child. The prepare handler is called before the processing of the fork subroutine commences. The parent handler is called after the processing of the fork subroutine completes in the parent process. The child handler is called after the processing of the fork subroutine completes in the child process.

Table 87. Execution scheduling

Name	Description
pthread_attr_setschedparam	Sets the value of the schedparam attribute of the specified thread attributes object. The given schedparam attribute specifies the scheduling parameters of a thread created with this attributes object.
pthread_attr_getschedparam	Gets the value of the schedparam attribute of the specified thread attributes object.
pthread_attr_setscope	The contention scope can only be set before thread creation by setting the contention-attribute of a thread attributes object. The pthread_attr_setscope subroutine sets the attribute to the specified value.
pthread_attr_getscope	Gets the contention-scope attribute of the specified thread attributes object.
pthread_attr_setinheritsched	Sets the inheritsched attribute of the specified thread attributes object to a given value.
pthread_attr_getinheritsched	Gets the inheritsched attribute of the specified thread attributes object.
pthread_attr_setschedpolicy	Sets the schedpolicy attribute of the specified thread attributes object.
pthread_attr_getschedpolicy	Gets the schedpolicy attribute of the specified thread attributes object.

Name	Description
pthread_setschedparam	Dynamically sets the schedpolicy and schedparam attributes of the specified thread. The given schedpolicy attribute specifies the scheduling policy of the thread. The given schedparam attribute specifies the scheduling parameters. The implementation of this subroutine is dependent on the priority scheduling POSIX option. The priority scheduling POSIX option is implemented in the operating system. If the target thread has system contention-scope, the process must have root authority to set the scheduling policy to either SCHED_FIFO or SCHED_RR.
pthread_getschedparam	Returns the current schedpolicy and schedparam attributes of the thread thread. The schedpolicy attribute specifies the scheduling policy of a thread.

Table 88. Synchronization

Name	Description
pthread_mutexattr_init	Initializes a mutex attributes object with the default value for all of the attributes defined by the implementation.
pthread_mutexattr_destroy	Destroys a mutex attributes object; the object becomes, in effect, uninitialized.
pthread_mutexattr_setpshared	Sets the process-shared attribute in a given initialized attributes object.
pthread_mutexattr_getpshared	Obtains the value of the process-shared attribute from the given attributes object.
pthread_mutexattr_setprioceiling pthread_mutexattr_getprioceiling pthread_mutexattr_setprotocol pthread_mutexattr_getprotocol pthread_mutex_setprioceiling pthread_mutex_getprioceiling	AIX does not support these routine; the symbols are provided but they always return ENOSYS.
pthread_mutex_init	Initializes the given mutex with attributes specified by a given attributes object. If the attributes object is NULL, the default mutex attributes are used.
pthread_mutex_destroy	Destroys the specified mutex object; the mutex object becomes, in effect, uninitialized.
pthread_mutex_lock	The specified mutex object is locked by calling. If the mutex is already locked, the calling thread blocks until the mutex becomes available.
pthread_mutex_trylock	Identical to pthread_mutex_lock, except that if the referenced mutex object is currently locked (by any thread, including the current thread), the call returns immediately.
pthread_mutex_unlock	Releases the referenced mutex object. The manner in which a mutex is released is dependent upon the mutex's type attribute.
pthread_condattr_init	Initializes a specified condition variable attributes object with the default value for all of the attributes defined by the implementation.
pthread_condattr_destroy	Destroys a specified condition variable attributes object; the object becomes, in effect, uninitialized.

Name	Description
pthread_condattr_setpshared	Sets the value of the pshared attribute of the specified condition attributes object. This attribute specifies the process sharing of the condition variable created with this attributes object.
pthread_condattr_getpshared	Returns the value of the pshared attribute of the specified condition attribute object. This attribute specifies the process sharing of the condition variable created with this attributes object.
pthread_cond_init	Initializes the given condition variable with attributes given by a condition attributes object. If that object is NULL, the default condition variable attributes are used.
pthread_cond_destroy	Destroys the given condition variable; the object becomes, in effect, uninitialized.
pthread_cond_wait	Blocks on a condition variable. Must be called with a specified mutex locked by the calling thread or undefined behavior will result.
pthread_cond_timedwait	Same as pthread_cond_wait except that an error is returned if the specified absolute time passes (that is, system time equals or exceeds the specified absolute time) before the specified condition is signaled or broadcasted, or if the absolute time specified has already been passed at the time of the call.
pthread_cond_signal	Unblocks one or more threads blocked on the specified condition.
pthread_cond_broadcast	Unblocks all the blocked threads on the specified condition.

Table 89. Thread specific data

Name	Description
pthread_key_create	Creates a thread-specific data key. The key is shared among all threads within the process, but each thread has specific data associated with the key. The thread-specific data is a void pointer, initially set to NULL. An optional destructor routine can be specified. It will be called for each thread when it is terminated and detached, after the call to the cleanup routines, if the specific value is not NULL.

Name	Description
pthread_key_delete	Deletes the given thread-specific data key, previously created with the pthread_key_create subroutine. The application must ensure that no thread-specific data is associated with the key.
pthread_setspecific	Associates a thread-specific value with a key obtained through a previous call to pthread_key_create. Different threads may bind different values to the same key.
pthread_getspecific	Returns the value currently bound to the specified key on behalf of the calling thread.
pthread_cleanup_push	Pushes the specified cancellation cleanup handler routine onto the calling thread's cancellation cleanup stack.
pthread_cleanup_pop	Removes the routine at the top of the calling thread's cancellation cleanup stack and optionally invokes it (if execute is non-zero).

10.1.2.4 UNIX 98 specification

For the sake of completeness, Table 90 lists additional pthread routines under AIX 5L that conform to the UNIX 98 specification. These routines are not part of POSIX, but part of the Single Unix Specification Interfaces.

Table 90. UNIX 98

Name	Description
pthread_rwlockattr_init	Initializes the read-write specified lock with the attributes referenced by the given read-write lock attribute object. If that object is NULL, the default read-write lock attributes are used.
pthread_rwlockattr_destroy	Destroys the specified read-write lock attribute object and releases any resources used by the lock.
pthread_rwlockattr_setpshared	Sets the process-shared attribute in the given initialized read-write lock attributes object.
pthread_rwlockattr_getpshared	Obtains the value of the process-shared attribute from the given initialized read-write lock attributes object.
pthread_rwlock_init	Initializes the specified read-write lock with the attributes from a given read-write lock attributes object. If that object is NULL, the default read-write lock attributes are used.

Name	Description
pthread_rwlock_destroy	Destroys the specified read-write lock object and releases any resources used by the lock.
pthread_rwlock_rdlock	Applies a read lock to the given read-write lock. The calling thread acquires the read lock if a writer does not hold the lock and there are no writers blocked on the lock.
pthread_rwlock_tryrdlock	Applies a read lock as in the pthread_rwlock_rdlock function with the exception that the function fails if any thread holds a write lock on the specified read-write lock or there are writers blocked the lock.
pthread_rwlock_wrlock	Applies a write lock to the given read-write lock. The calling thread acquires the write lock if no other thread (reader or writer) holds the read-write lock. Otherwise, the thread blocks (that is, does not return from the pthread_rwlock_wrlock call) until it can acquire the lock.
pthread_rwlock_trywrlock	Applies a write lock like the pthread_rwlock_wrlock function, with the exception that the function fails if any thread currently holds the specified read-write lock (for reading or writing).
pthread_rwlock_unlock	Releases a lock held on the specified read-write lock object.
pthread_setconcurrency	Allows an application to inform the threads implementation of its desired concurrency level. The actual level of concurrency provided by the implementation as a result of this function call is unspecified.
pthread_getconcurrency	Returns the value set by a previous call to the pthread_setconcurrency function.
pthread_attr_setguardsize	Sets the guardsize attribute in a thread attribute object. The guardsize attribute controls the size of the guard area for the created thread's stack. The guardsize attribute provides protection against overflow of the thread’s stack pointer.
pthread_attr_getguardsize	Gets the guardsize attribute of a thread attributes object.
pthread_mutexattr_settype	Sets the mutex type attribute of a mutex attributes object to a given type.

Name	Description
pthread_mutexattr_gettype	Gets the type attribute of a given mutex attributes object.
pthread_suspend	Suspends execution of specified thread.
pthread_continue	Resumes execution of specified thread.

10.1.2.5 Thread extensions - _np routines

For the sake of completeness, this section lists pthread routines under AIX 5L which are “non-portable.” They all have the extension _np and should be avoided, because it would, in general, render the code non-portable. The routines are not part of the POSIX threads, but provide compatibility with DCE threads (implementation of POSIX 1003 Draft 4). The routines are shown in Table 91.

Table 91. Non-portable thread routines in AIX 5L

Routine	Routine (continued)
pthread_atfork_np	pthread_getthrds_np
pthread_atfork_unregister_np	pthread_getunique_np
pthread_attr_getsuspendstate_np	pthread_join_np
pthread_attr_setstacksize_np	pthread_lock_global_np
pthread_attr_setsuspendstate_np	pthread_mutexattr_getkind_np
pthread_cleanup_information_np	pthread_mutexattr_setkind_np
pthread_cleanup_pop_np	pthread_set_mutexattr_default_np
pthread_cleanup_push_np	pthread_setcancelstate_np
pthread_clear_exit_np	pthread_signal_to_cancel_np
pthread_continue_np	pthread_suspend_np
pthread_continue_others_np	pthread_suspend_others_np
pthread_delay_np	pthread_test_exit_np
pthread_get_expiration_np	pthread_unlock_global_np
pthread_getrusage_np	

10.2 Thread scheduling

When porting threaded applications, it is important to understand which thread models are implemented in AIX 5L. For instance, the choice of thread model affects the semantics of a thread’s scheduling priority and policy.

Under the *system contention model*, the user thread’s underlying kernel thread will be scheduled against all other threads in the system.

Under the *process contention model*, the user thread will be scheduled by a POSIX pthread library scheduler against all other process contention model threads of the given process.

In the first case, the policy and priority is interpreted globally on the system; in the latter, they are being interpreted within a process.

10.2.1 Lightweight processes

User threads, or just threads, are mapped to underlying kernel threads. A *thread model* refers to the way this mapping is done. A *lightweight process* (LWP) refers to a kernel thread.

There are three possible thread models, corresponding to three different ways to map user threads to kernel threads:

- 1:1 model
- M:1 model
- M:N model

The mapping of user threads to kernel threads is done using virtual processors. A *virtual processor* (VP) is a library entity that is usually implicit. For a user thread, the virtual processor behaves as a CPU for a kernel thread. In the library, the virtual processor is a kernel thread or a structure bound to a kernel thread. That is, user threads sit on top of virtual processors which are themselves on top of kernel threads.

10.2.2 Bound thread scheduling

The 1:1 model is sometimes referred to as *bound thread scheduling*. In this model, each user thread is bound to a VP and linked to exactly one kernel thread (LWP). The VP is not necessarily bound to a real CPU (unless binding to a processor was done). Each VP can be thought of as a virtual CPU available for executing user code and system calls. A thread which is bound to a VP is said to have *system scope* because it is directly scheduled with all the other user threads by the kernel scheduler. Most of the user threads programming facilities are directly handled by the kernel threads. Figure 62 on page 325 illustrates this model.

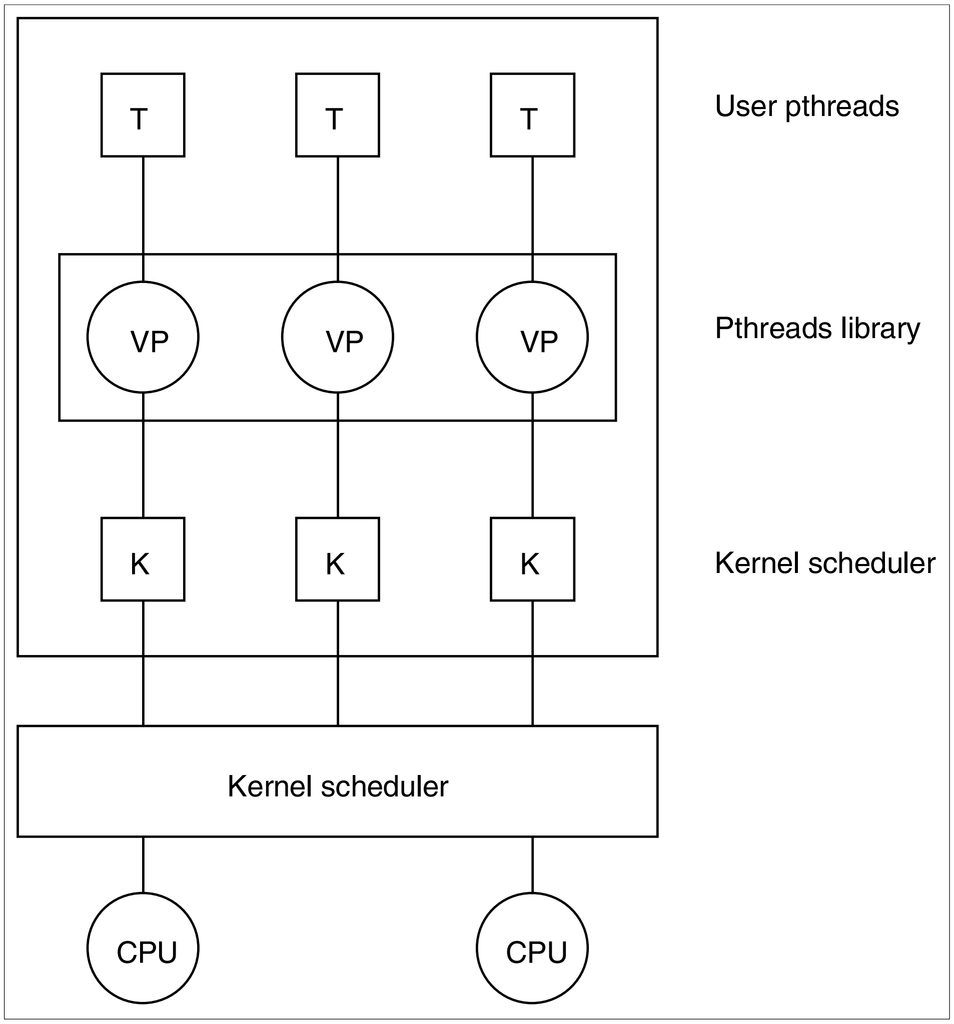


Figure 62. 1:1 thread model

10.2.3 Multiplexed thread scheduling

Multiplexed thread scheduling refers to the case where threads share a pool of available LWPs.

In the M:1 model, all user threads run on one VP and are linked to exactly one LWP; the mapping is handled by a POSIX thread library scheduler. All user threads programming facilities are completely handled by the library. This model can be used on any system, especially on traditional single-threaded systems. Figure 63 on page 326 illustrates this model.

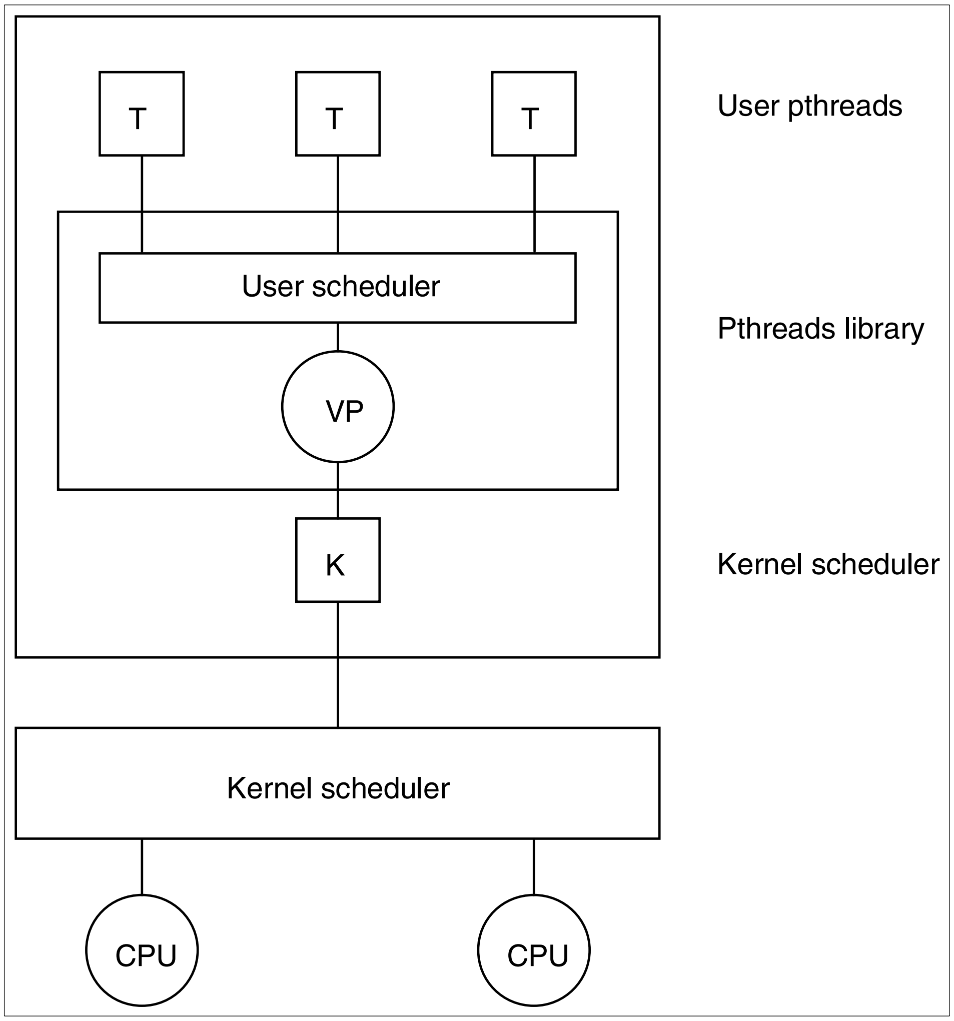


Figure 63. M:1 thread model

In the M:N model, several user threads can share the same virtual processor or the same pool of VPs. A thread that is not bound to a VP is said to be a local or *process scope*, because it is not directly scheduled with all the other threads by the kernel scheduler. The pthreads library will handle the scheduling of user threads to the VP and then the kernel will schedule the associated kernel thread. This is the most efficient and most complex thread model; the user threads programming facilities are shared between the threads library and the kernel threads. Figure 64 on page 327 illustrates this model.

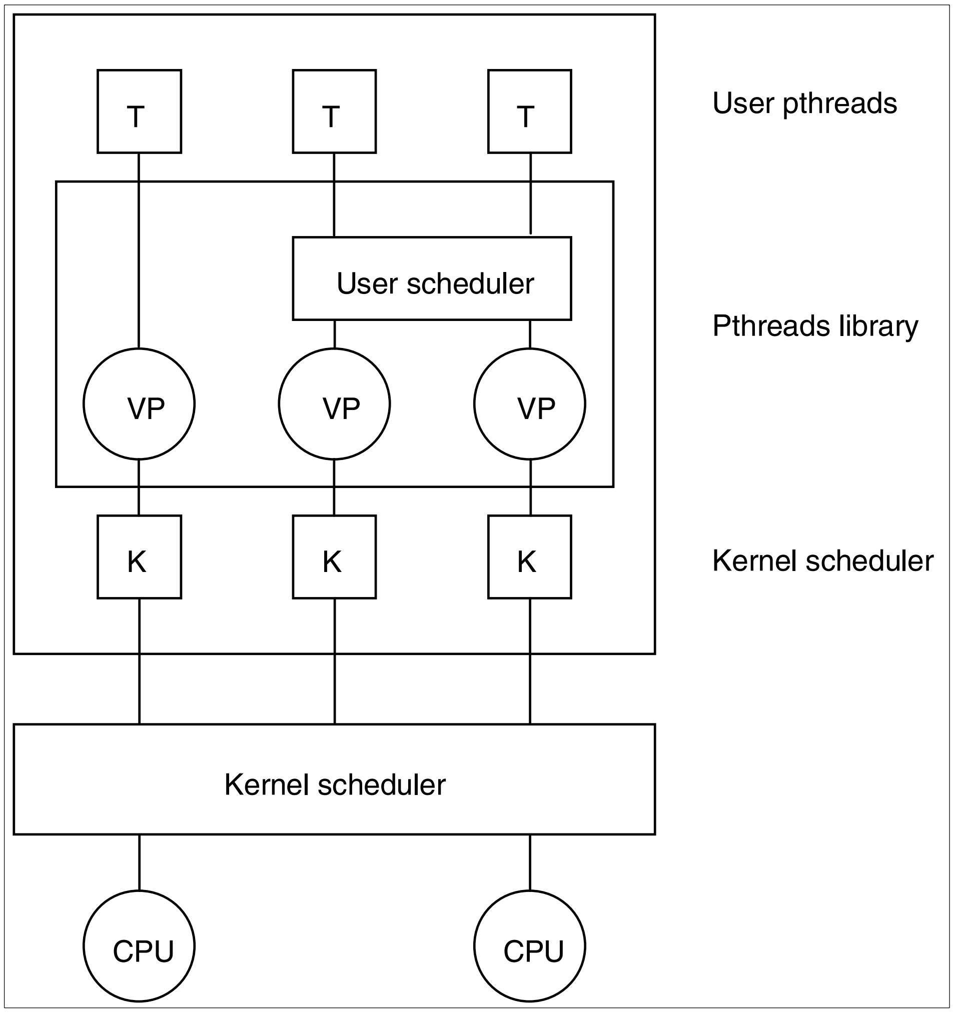


Figure 64. M:N thread model

Table 103 on page 379 lists the supported thread models for various UNIX implementations.

10.2.4 Comparing bound and multiplexed threads

Bound thread scheduling differs from multiplexed thread scheduling in the following important ways:

- A bound thread is permanently associated to its kernel thread. Hence, it is exempt from the intermediate level of scheduling provided by the POSIX threads library used under the M:1 and M:N thread models.

- A bound thread executes exactly when its associated kernel thread is scheduled by the kernel scheduler.
- A bound thread's scheduling policy is related to the underlying kernel thread. In AIX 5L, only kernel threads with root authority can use a fixed-priority scheduling policy.

Multiplexed thread scheduling has the following important characteristic:

- A multiplexed thread is subject to two levels of scheduling. First, the thread is assigned to a kernel thread and preempted by a POSIX thread library scheduler. Second, the kernel scheduler assigns the LWPs to processors and then preempts them.

10.2.5 Scheduling scope, policy, and priority

The pthreads library allows the programmer to control the execution scheduling of the threads. The control can be performed in two different ways:

- By setting scheduling attributes when creating a thread.
- By dynamically changing the scheduling attributes of a created and executing thread.

A thread has three scheduling parameters:

Scope	The contention scope of a thread is defined by the thread model used in the threads library.
Policy	The scheduling policy of a thread defines how the scheduler treats the thread once it gains control of the CPU.
Priority	The scheduling priority of a thread defines the relative importance of the work being done by each thread.

In general, controlling the scheduling parameters of threads is important only for threads that are "compute intensive." Thus, the threads library provides default values that are sufficient for most cases.

Controlling the scheduling of a thread is often a complicated task. Because the scheduler can handle all threads system or process-wide, depending on the scope context, the scheduling parameters of a thread can interact with those of all other threads in the process and in the other processes on the system.

10.2.5.1 Scope

The contention scope can only be set before thread creation by setting the contention-scope attribute of a thread attributes object. If no specific value is chosen, the default choice is the process scope.

10.2.5.2 Policies

On AIX 5L, the threads library provides three scheduling policies:

SCHED_FIFO	First-in first-out (FIFO) scheduling. Each thread has a fixed priority; when multiple threads have the same priority level, they run to completion in FIFO order.
SCHED_RR	Round-robin (RR) scheduling. Each thread has a fixed priority; when multiple threads have the same priority level, they run for a fixed time slice in FIFO order.
SCHED_OTHER	Default AIX scheduling. Each thread has an initial priority that is dynamically modified by the scheduler according to the thread's activity; thread execution is time-sliced. On other systems, this scheduling policy may be different.

Normally, applications should use the default scheduling policy unless a specific application requires the use of a fixed-priority scheduling policy.

Using the RR policy ensures that all threads having the same priority level will be scheduled equally, regardless of their activity. This can be useful in programs where threads have to read sensors or write actuators.

Using the FIFO policy should be done with great care. A thread running with FIFO policy runs to completion unless it is blocked by some calls, such as performing input and output operations. A high-priority FIFO thread may not be preempted and can affect the global performance of the system. For example, threads doing intensive calculations, such as inverting a large matrix, should never run with FIFO policy.

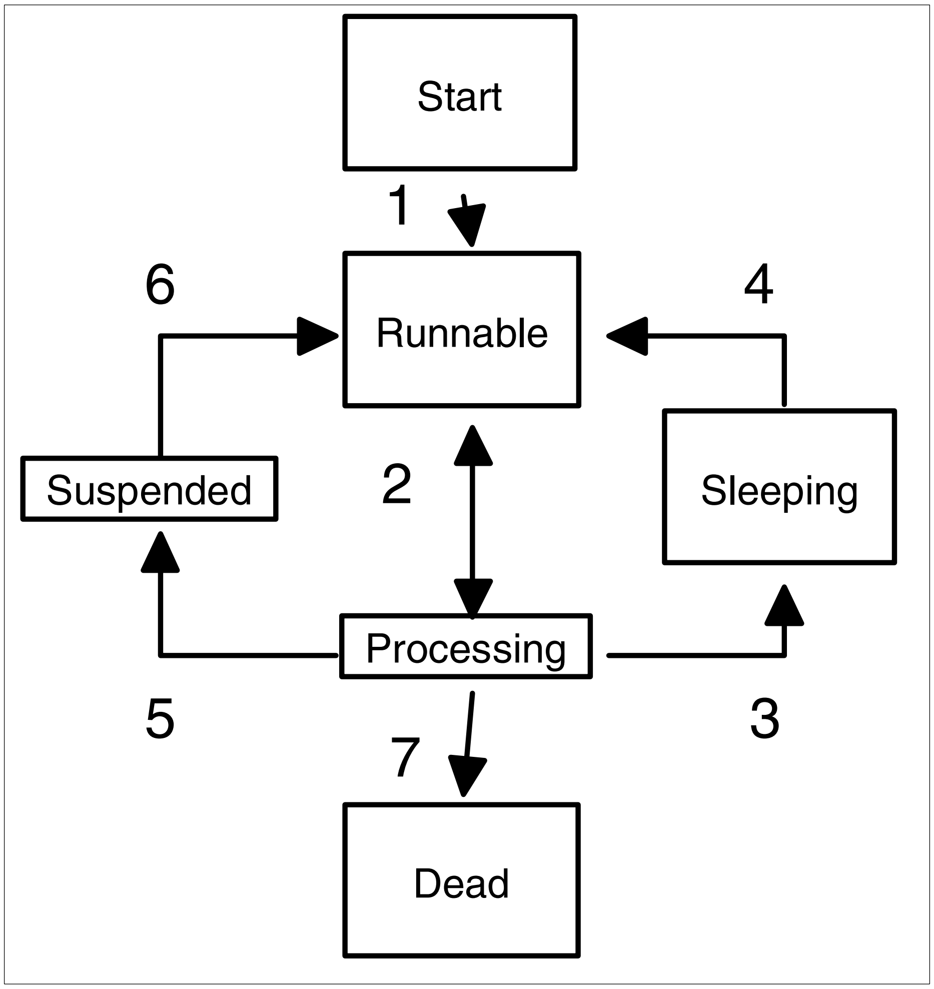


Figure 65. State transitions for a common multiplexed thread

The multiplexed threads run over a state machine, as shown in Figure 65. The state transitions are described as follows:

1. At creation time, the system initializes the thread in the RUNNABLE state.
2. When it is mapped to a kernel LWP from the pool, it transitions from RUNNABLE to PROCESSING state when the kernel dispatches the LWP for execution. While in the PROCESSING state, the thread issues kernel calls and remains mapped to the LWP. In the same way, if the kernel call blocks the multiplexed thread, then the LWP will also block. In the next

piece of code, the multiplexed thread and its associated LWP will block until the read request completes:

```
read (file_description, buffer, size);
```

3. If, during the processing time, the thread blocks waiting for a synchronization event, described in the next section, it goes to the SLEEPING state. In the SLEEPING state, it is no longer mapped to a LWP.
4. When a signal wakes up the thread, it then transitions from SLEEPING to the RUNNABLE state again.
5. It is also possible for a thread to transition to the SUSPENDED state. It remains there until another thread from the user level resumes it.
6. At the finalization time, the thread transitions from PROCESSING to the DEAD state, when it releases its resources. The system will remove the threads data on DEAD state from the process data space.

10.2.5.3 Priority

The priority is an integer value, in the range from 1 to 127. 1 is the least-favored priority, 127 is the most-favored. Priority level 0 cannot be used: it is reserved for the system. Note that in AIX 5L, the kernel inverts the priority levels. For the AIX 5L kernel, the priority is in the range from 0 to 255, where 0 is the most favored priority and 255 the least-favored. Commands, such as the `ps -emo THREAD` command, report the kernel priority.

The threads library handles the priority through a sched_param structure, defined in the sys/sched.h header file. Currently, this structure contains two fields:

sched_priority	Specifies the priority.
sched_policy	This field is ignored by the threads library and should not be used.

The scheduling policy can be set when creating a thread by setting the schedpolicy attribute of the thread attributes object. In the following code fragment, a thread is created with the round-robin scheduling policy, using a priority level of 3:

```
sched_param schedparam;
schedparam.sched_priority = 3;
pthread_attr_init(&attr);
pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&attr, SCHED_RR);
pthread_attr_setschedparam(&attr, &schedparam);
pthread_create(&thread, &attr, &start_routine, &args);
```

```
pthread_attr_destroy(&attr);
```

The scheduling policy can also be altered during execution of a thread. The current schedpolicy and schedparam attributes of a thread are returned by the pthread_getschedparam subroutine. These attributes can be set by calling the pthread_setschedparam subroutine. If the target thread is currently running on a processor, the new scheduling policy and priority will be implemented the next time the thread is scheduled. If the target thread is not running, it may be scheduled immediately at the end of the subroutine call.

10.2.6 Porting issues

In general, applications should use the default scheduling policy, unless a specific application requires the use of a fixed-priority scheduling policy.

Depending on the type of application, the administrator can choose to use a different thread model. Tests have shown that certain applications can perform much better with the 1:1 model. This is an important point because the default thread model is M:N. By simply setting the environment variable AIXTHREAD_SCOPE=S for that process, we can set the thread model to 1:1 and then compare the performance to its previous performance when the thread model was M:N.

If you see an application creating and deleting threads, it could be due to kernel threads being harvested because of the 8:1 default ratio of user threads to kernel threads. This harvesting, along with the overhead of the library scheduling, can affect the performance. On the other hand, when thousands of user threads exist, there may be less overhead to schedule them in user space in the library rather than manage thousands of kernel threads. You should always try changing the scope if you encounter a performance problem when using pthreads; in many cases, the system scope can provide better performance.

Material regarding performance and tuning can be found in Section 10.8, “Tuning” on page 367.

10.3 Thread creation, termination, and synchronization

This section briefly discusses the process of creating and working with POSIX phtreads on AIX 5L.

10.3.1 Creating threads

A thread has attributes, which specify the characteristics of the thread. The attributes default values fit for most common cases. To control thread attributes, a thread attributes object must be defined before creating the thread.

10.3.1.1 Thread attributes object

The thread attributes are stored in an opaque object, the thread attributes object, used when creating the thread. It contains several attributes, depending on the implementation of POSIX options. It is accessed through a variable of type pthread_attr_t. In AIX 5L, the pthread_attr_t data type is a pointer to a structure; on other systems, it may be a structure or another data type.

The thread attributes object is initialized to default values by the pthread_attr_init subroutine; see Table 92.

Table 92. Attributes of the pthread_attr_t type for AIX 5L

Attribute	Default value
Detachstate	PTHREAD_CREATE_JOINABLE
Contention-scope	PTHREAD_SCOPE_PROCESS (the default ensures compatibility with implementations that do not support this POSIX option).
Inheritsched	PTHREAD_INHERIT_SCHED
Schedparam	A sched_param structure where the sched_prio field is set to 1, the least favored priority.
Schedpolicy	SCHED_OTHER
Stacksize	PTHREAD_STACK_MIN
Guardsize	PAGESIZE

The thread attributes object is destroyed by the pthread_attr_destroy subroutine. Here is an example:

```
#include <pthread.h>            /* must be the first #include file */
...
pthread_attr_t attr;            /* defines a variable somewhere */
                                /* in the code, globally or */
                                /* locally. */
...
pthread_attr_init(&attr);       /* creates and initializes with */
                                /* default variables, used for */
```

```
.....                             /* setting non-default values */
pthread_attr_destroy(&attr);      /* releases the variable */
```

Setting other attribute values can be done using the pthread_attr_set... routines mentioned in Section 10.1, “Introduction to threads” on page 307. For example, the detachstate attribute can hold one of two values:

PTHREAD_CREATE_DETACHED	Specifies that the thread will be created in the detached state.
PTHREAD_CREATE_JOINABLE	Specifies that the thread will be created in the joinable state.

For more details on manipulation of the attributes, please consult the online documentation.

The same attributes object can be used to create several threads. It can also be modified between two thread creations. When the threads are created, the attributes object can be destroyed without affecting the threads created with it.

10.3.1.2 Example of the pthread_create routine

The creation of a new thread is performed using the pthread_create subroutine. This function creates a new thread and makes it runnable. It is defined as:

```
int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void
*(*start_routine) (void), void *arg);
```

where the arguments are:

thread	A pointer to the new thread’s ID variable.
attr	Points to a pthread_attr_t variable properly declared and initialized.
start_routine	A pointer to the routine that will be executed by the new thread.
arg	A pointer to arguments that will be passed to the new thread.

When calling the pthread_create subroutine, you may specify a thread attributes object. If you specify a NULL pointer, the created thread will have the default attributes. Thus, the code fragment:

```
pthread_t thread;
pthread_attr_t attr;
...
pthread_attr_init(&attr);
```

```
pthread_create(&thread, &attr, start_routine, NULL);
pthread_attr_destroy(&attr);
```

is equivalent to:

```
pthread_t thread;
...
pthread_create(&thread, NULL, start_routine, NULL);
```

When calling the pthread_create subroutine, you must specify an entry-point routine. This routine, provided by your program, is similar to the main routine for a process. It is the user routine executed by the new thread. When the thread returns from this routine, the thread is automatically terminated.

The entry-point routine has one parameter, a void pointer, specified when calling the pthread_create subroutine. You may specify a pointer to some data, such as a string or a structure. The creating thread (the one calling the pthread_create subroutine) and the created thread must agree upon the actual type of this pointer. The entry-point routine returns a void pointer. After the thread termination, this pointer is stored by the threads library unless the thread is detached.

The thread ID of a newly created thread is returned to the creating thread through the thread parameter. A thread ID is an opaque object; its type is pthread_t. In AIX 5L, the pthread_t data type is an integer. On other systems, it may be a structure, a pointer, or any other data type.The caller can use this thread ID to perform various operations on the thread.

Depending on the scheduling parameters, the new thread may start running before the call to the pthread_create subroutine returns. It may even happen that, when the pthread_create subroutine returns, the new thread has already terminated. The thread ID returned by the pthread_create subroutine through the thread parameter is then already invalid. It is, therefore, important to check for the ESRCH error code returned by threads library subroutines when using a thread ID as a parameter.

If the pthread_create subroutine is unsuccessful, no new thread is created; the thread ID in the thread parameter is invalid, and the appropriate error code is returned.

10.3.2 Termination of threads

Execution of a thread can end in several ways.

10.3.2.1 Exiting

A process can exit at any time from any thread by calling the exit subroutine. Similarly, a thread can exit at any time by calling the pthread_exit subroutine.

Calling the exit subroutine terminates the entire process, including all its threads. In a multithreaded program, the exit subroutine should only be used when the entire process needs to be terminated; for example, in the case of an unrecoverable error. The pthread_exit subroutine should be preferred, even for exiting the initial thread.

Calling the pthread_exit subroutine terminates the calling thread. The status parameter is saved by the library and can be further used when joining (explained in Section 10.3.3, "Joining threads" on page 343) the terminated thread. Calling the pthread_exit subroutine is similar, but not identical, to returning from the thread's initial routine. The result of returning from the thread's initial routine depends on the thread:

- Returning from the initial thread implicitly calls the exit subroutine, thus terminating all the threads in the process.
- Returning from another thread implicitly calls the pthread_exit subroutine. The return value has the same role as the status parameter of the pthread_exit subroutine.

It is recommended always to use the pthread_exit subroutine to exit a thread to avoid implicitly calling the exit subroutine.

Exiting the initial thread by calling the pthread_exit subroutine does not terminate the process; it only terminates the initial thread. If the initial thread is terminated, the process will be terminated when the last thread in it terminates. In this case, the process return code (usually the return value of the main routine or the parameter of the exit subroutine) is 0 if the last thread was detached or 1 otherwise.

Note

It is important to note that the pthread_exit subroutine frees any thread-specific data, including the thread's stack. Any data allocated on the stack becomes invalid, since the stack is freed and the corresponding memory may be reused by another thread. Therefore, thread synchronization objects (mutexes and condition variables) allocated on a thread's stack must be destroyed before the thread calls the pthread_exit subroutine.

Unlike the exit subroutine, the pthread_exit subroutine does not clean up system resources shared among threads. For example, files are not closed by the pthread_exit subroutine, since they may be used by other threads.

10.3.2.2 Killing a thread

It is possible to send a signal to a specific thread. The routine pthread_kill is described in Section 10.5.3, “Signal generation” on page 355.

10.3.2.3 Canceling threads

The thread cancellation mechanism allows a thread to terminate the execution of any other thread in the process in a controlled manner. The target thread (that is, the one that's being canceled) can hold cancellation requests pending in a number of ways and perform application-specific cleanup processing when the notice of cancellation is acted upon. When canceled, the thread implicitly calls the pthread_exit((void *)-1) subroutine. The cancellation of a thread is requested by calling the pthread_cancel subroutine. When the call returns, the request has been registered, but the thread may still be running.

The cancelability state and type of a thread determines the action taken upon receipt of a cancellation request:

Disabled cancelability	Any cancellation request is set pending, until the cancelability state is changed or the thread is terminated in another way. A thread should disable cancelability only when performing operations that cannot be interrupted. For example, if a thread is performing some complex file save operations (such as an indexed database) and is canceled during the operation, the files may be left in an inconsistent state. To avoid this, the thread

	should disable cancelability during the file save operations.
Deferred cancelability	Any cancellation request is set pending until the thread reaches the next cancellation point. This is the default cancelability state. This cancelability state ensures that a thread can be cancelled, but limits the cancellation to specific moments in the thread's execution, called cancellation points. A thread canceled on a cancellation point leaves the system in a safe state; however, user data may be inconsistent or locks may be held by the canceled thread. To avoid these situations, you may use cleanup handlers or disable cancelability within critical regions.
Asynchronous cancelability	Any cancellation request is acted upon immediately. A thread that is asynchronously canceled while holding resources may leave the process, or even the system, in a state from which it is difficult or impossible to recover.

Cancellation points are points inside of certain subroutines where a thread must act on any pending cancellation request if deferred cancelability is enabled. An explicit cancellation point can also be created by calling the pthread_testcancel subroutine. This subroutine simply creates a cancellation point. If deferred cancelability is enabled, and if a cancellation request is pending, the request is acted upon, and the thread is terminated. Otherwise, the subroutine simply returns.

Other cancellation points occur when calling the following subroutines:

- pthread_cond_wait
- pthread_cond_timedwait
- pthread_join

The following code shows a thread where the cancelability is disabled for a set of instructions and then restored using the pthread_setcancelstate function:

```
void *Thread(void *string)
{
   int i;
   int o_state;

   /* disables cancelability */
   pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &o_state);

   /* writes five messages */
   for (i=0; i<5; i++)
      printf("%s\n", (char *)string);

   /* restores cancelability */
   pthread_setcancelstate(o_state, &o_state);

   /* writes further */
   while (1)
      printf("%s\n", (char *)string);
   pthread_exit(NULL);
}
```

Table 93 lists the functions that contain cancellation points, as required in the POSIX threads standard.

Table 93. Cancellation point functions

Name	Name (continued)	Name (continued)
aio_suspend	close	creat
fcntl	fsync	getmsg
getpmsg	lockf	mq_receive
mq_send	msgrcv	msgsnd
msync	nanosleep	open
pause	poll	pread
pthread_cond_timedwait	pthread_cond_wait	pthread_join
pthread_testcancel	putpmsg	pwrite
read	readv	select
sem_wait	sigpause	sigsuspend
sigtimedwait	sigwait	sigwaitinfo
sleep	system	tcdrain
usleep	wait	wait3
waitid	waitpid	write

Name	Name (continued)	Name (continued)
writev		

Table 94 lists the functions that *may* contain cancellation points, as specified in the POSIX threads standard.

Table 94. Function where cancellation points may occur

Name	Name (continued)	Name (continued)
catclose	catgets	catopen
closedir	closelog	ctermid
dbm_close	dbm_delete	dbm_fetch
dbm_nextkey	dbm_open	dbm_store
dlclose	dlopen	endgrent
endpwent	endutxent	`fclose`
fcntl	fflush	fgetc
fgetpos	fgets	fgetwc
fgetws	fopen	fprintf
fputc	fputs	fputwc
fputws	fread	freopen
fscanf	`fseek`	fseeko
fsetpos	ftell	`ftello`
ftw	fwprintf	fwrite
fwscanf	getc	getc_unlocked
getchar	getchar_unlocked	getcwd
getdate	getgrent	getgrgid
getgrgid_r	getgrnam	getgrnam_r
getlogin	getlogin_r	getpwent
getpwnam	getpwnam_r	getpwuid
getpwuid_r	gets	getutxent
getutxid	getutxline	getw

Name	Name (continued)	Name (continued)
getwc	getwchar	getwd
glob	iconv_close	iconv_open
ioctl	lseek	mkstemp
nftw	opendir	openlog
pclose	perror	popen
printf	putc	putc_unlocked
putchar	putchar_unlocked	puts
pututxline	putw	putwc
putwchar	readdir	readdir_r
remove	rename	rewind
rewinddir	scanf	seekdir
semop	setgrent	setpwent
setutxent	strerror	syslog
tmpfile	tmpnam	ttyname
ttyname_r	ungetc	ungetwc
unlink	vfprintf	vfwprintf
vprintf	vwprintf	wprintf
wscanf		

10.3.2.4 Cleanup handlers

Cleanup handlers are specific to each thread. A thread can have several cleanup handlers; cleanup handlers are stored in a thread-specific LIFO (Last In First Out) stack. They are all called in the following cases:

- The thread returns from its entry-point routine.
- The thread calls the pthread_exit subroutine.
- The thread acts on a cancellation request.

A cleanup handler is pushed onto the cleanup stack by the pthread_cleanup_push subroutine. The pthread_cleanup_pop subroutine pops the topmost cleanup handler from the stack, and optionally executes it. Use this subroutine when the cleanup handler is no longer needed.

The cleanup handler is a user-defined routine. It has one parameter, a void pointer, specified when calling the pthread_cleanup_push subroutine. You may specify a pointer to some data the cleanup handler needs to perform its operation.

In the following example, a buffer is allocated for performing some operation. With deferred cancelability enabled, the operation may be stopped at any cancellation point. A cleanup handler is established to free the buffer in that case.

```
/* the cleanup handler */

cleaner(void *buffer)

{
        free(buffer);
}

/* fragment of another routine */
...
myBuf = malloc(1000);
if (myBuf != NULL) {

        pthread_cleanup_push(cleaner, myBuf);

        /*
         *       perform any operation using the buffer,
         *       including calls to other functions
         *       and cancellation points
         */

        /* pops the handler and frees the buffer in one call */
        pthread_cleanup_pop(1);
}
```

Using deferred cancelability ensures that the thread will not act on any cancellation request between the buffer allocation and the registration of the cleanup handler, because neither the malloc subroutine nor the pthread_cleanup_push subroutine provides any cancellation point. When popping the cleanup handler, the handler is executed, freeing the buffer. More complex programs may not execute the handler when popping it, because the cleanup handler should be thought of as an emergency exit for the protected portion of code.

10.3.2.5 Balancing the push and pop operations

The pthread_cleanup_push and pthread_cleanup_pop subroutines should always appear in pairs within the same lexical scope, that is, within the same function and the same statement block. They can be thought of as left and right parentheses enclosing a protected portion of code.

The reason for this rule is that on some systems, these subroutines are implemented as macros. The pthread_cleanup_push subroutine is implemented as a left brace, followed by other statements:

```
#define pthread_cleanup_push(rtm,arg) { \
        /* other statements */
```

The pthread_cleanup_pop subroutine is implemented as a right brace following other statements:

```
#define pthread_cleanup_pop(ex) \
        /* other statements */  \
}
```

10.3.3 Joining threads

Joining a thread means waiting for it to terminate. The pthread_join subroutine provides a simple mechanism that allows a thread to wait for another thread to terminate.

The pthread_join subroutine blocks the calling thread until the specified thread terminates. The target thread (the thread whose termination is awaited) must not be detached. If the target thread is already terminated, but not detached, the pthread_join subroutine returns immediately. Once a target thread has been joined, it is automatically detached, and its storage can be reclaimed.

Table 95 indicates the two possible cases when a thread calls the pthread_join subroutine, depending on the state and the detachstate attribute of the target thread.

Table 95. Effect of calling pthread_join

Status of target	Undetached target	Detached target
Target is still running.	The caller is blocked until the target is terminated.	The call returns immediately, indicating an error.
Target is terminated.	The call returns immediately, indicating a successful completion.	The call returns immediately, indicating an error.

It is possible for several threads to join the same target thread if the target is not detached. The success of this operation depends on the order of the calls to the pthread_join subroutine and the moment when the target thread terminates.

- Any call to the pthread_join subroutine occurring before the target thread's termination blocks the calling thread.
- When the target thread terminates, all blocked threads are awoken, and the target thread is automatically detached.
- Any call to the pthread_join subroutine occurring after the target thread's termination will fail, because the thread is detached by the previous join.
- If no thread called the pthread_join subroutine before the target thread's termination, the first call to the pthread_join subroutine will return immediately, indicating a successful completion, and any further call will fail.

A thread cannot join itself; a deadlock would occur and it is detected by the library. However, two threads may try to join each other; they will deadlock. This situation is not detected by the library.

The pthread_join subroutine also allows a thread to return information to another thread. When a thread calls the pthread_exit subroutine or when it returns from its entry-point routine, it returns a pointer. This pointer is stored as long as the thread is not detached, and the pthread_join subroutine can return it.

10.3.4 Porting issues

To enhance the portability of programs using the threads library, the thread ID should always be handled as an opaque object. For this reason, thread IDs should be compared using the pthread_equal subroutine. Never use the C equality operator (==) because the pthread_t data type may be neither an arithmetic type nor a pointer.

In AIX 5L, the pthread_cleanup_push and pthread_cleanup_pop subroutines are library routines, and can be unbalanced within the same statement block. However, they must be balanced in the program, since the cleanup handlers are stacked.

Not following the balancing rule for the pthread_cleanup_push and pthread_cleanup_pop subroutines may lead to compile errors or to unexpected behavior of your programs when porting to other systems.

10.4 Synchronized access to data objects

Access to shared data presents potential problems when it is shared between processes and when it is shared within a process by multiple threads.

10.4.1 Synchronization

Synchronization is a programming method that allows multiple threads to coordinate their data accesses, therefore avoiding the situation where one thread can change a piece of data at the same time another one is reading or writing the same piece of data. This situation is commonly called a *race condition.*

Consider, for example, a single counter X that is incremented by two threads, A and B. If X is originally 1, then by the time threads A and B increment the counter, X should be 3. Both threads are independent entities and have no synchronization between them. Although the C statement X++ looks simple enough to be atomic, the generated assembly code may not be, as shown in the following pseudo-assembler code:

```
move X, REG    /* put the value of X on register   */
inc REG        /* increment register               */
move REG, X    /* store register value at X        */
```

If both threads are executed concurrently on two CPUs, or if the scheduling makes the threads alternatively execute on each instruction, the following steps may occur:

1. Thread A executes the first instruction and puts X, which is 1, into the thread A register. Then thread B executes and puts X, which is 1, into the thread B register.
2. Thread A executes the second instruction and increments the content of its register to 2. Then, thread B increments its register to 2. Nothing is moved to memory X, so memory X stays the same.
3. Thread A moves the content of its register, which is now 2, into memory X. Then thread B moves the content of its register, which is also 2, into memory X, overwriting thread A's value.

Note that, in most cases, thread A and thread B will execute the three instructions one after the other, and the result would be 3, as expected. Race conditions are usually difficult to discover because they occur intermittently.

To avoid this race condition, each thread should lock the data before accessing the counter and updating memory X. For example, if thread A takes a lock and updates the counter, it leaves memory X with a value of 2.

Once thread A releases the lock, thread B takes the lock and updates the counter, taking 2 as its initial value for X and incrementing it to 3, the expected result.

Basically, there are two ways for implementing how a thread can deal with the situation where it is trying to lock some data that is, in fact, already locked by another thread:

busy/wait This approach is based on the hope that the lock data will be available in a very short period of time. Basically, the thread enters a loop and continuously attempts to get the lock for the data. This model, despite its simplicity, normally runs well on multiple CPU machines. Otherwise, on a single CPU machine, the thread keeps occupying the CPU, and there is no chance for the other thread, that actually has the lock, to resume execution and free the locked data. This type of lock is also known as a spin lock.

sleep/wait This model is a bit more elaborate but still simple and understandable. The idea is to put the thread in a SLEEPING state while the required lock is not available. The operating system will reactivate the thread whenever the desired locked data is ready. On SLEEPING state, the thread is not mapped to any LWP and is not consuming CPU, which gives the opportunity for the other threads to run.

We cannot say that one implementation approach is better than the other. The decision is dependent on the problem and on the machine where it will run, and must be carefully considered at design time.

Another very important issue when dealing with multithreaded programs is the deadlock situation. This is a situation where a thread, for example, locked data and then attempts to re-lock it before unlocking. Another well-known situation of deadlock is when a thread, for example, th_a, locks a data called A and then attempts to lock the data B; but at the same time, another thread, th_b, locked the data B and then starts trying to lock the A data. It is a recursive interaction and results in an infinite deadlock.

The pthread library provides a set of synchronization primitives and its associated API. Respecting the formalism of those primitives is a very good way to avoid conflicts when using shared resources. The following section describes, in detail, the most common synchronization methods.

Note

The file descriptors in a process are shared by all of its threads. This can potentially generate data inconsistency when two or more threads are accessing the same file. In this kind of application, a lock mechanism must be implemented, and the file pointer should be tracked by each thread.

10.4.2 Mutex

The mutual exclusion lock (*mutex*) is the simplest synchronization primitive provided by the pthread library, and many of the other synchronization primitives are built upon it.

It is based on the concept of a resource that only one person can use in a period of time, for example, a chair or a pencil. If one person sits in a chair, no one can sit on it until the first person stands up. A mutex can only be *locked* by one thread and it can only be *unlocked* by that very same thread.

This kind of primitive is quite useful for creating critical sections. A *critical section* is a portion of code that must run atomically because it normally is handling resources, such as file descriptors, I/O devices, or shared data. A critical section is therefore delimited by the instructions that lock and unlock a mutex variable. This ensures that, at most, one thread is executing any critical section protected by the given mutex.

Ensuring that all threads acting on the same resource or shared data obey this rule is a very good practice to avoid trouble when programming with threads. The following program shows a very simple example that complies with this rule:

```
#include <pthread.h>                       /* include file for pthreads     */
#include <stdio.h>                         /* include file for printf()     */
#define num_threads 10;                    /* define the number of threads  */
main()                                     /* the main thread               */
{
   pthread_t th[ num_threads];             /* creates an array for threads  */
   pthread_mutex_t mutex;                  /* defines a mutex variable      */
   int i;
   ...                                     /* do other stuff                */
   pthread_mutex_init(&mutex, NULL);       /* creates the mutex             */
   for (i = 0; i < num_thrads; i++)        /* loop to create threads        */
      pthread_create(&th[i], NULL, thread_func, NULL);
   ...                                     /* do other stuff                */
   pthread_mutex_destroy(&mutex);          /* destroys the mutex            */
}

void * thread_func( void *)           /* the request handling thread   */
{
   pthread_mutex_lock(&mutex);        /* locks the mutex               */
   ...                                /* do all the work               */
   pthread_mutex_unlock(&mutex);      /* unlocks the mutex             */
   pthread_exit( NULL);               /* finishes the thread           */
}
```

In AIX 5L, mutexes cannot be re-locked by the same thread. This may not be the case on other systems. To enhance portability of your programs, assume that the following code fragment will result in a deadlock situation:

```
pthread_mutex_lock(&mutex);
pthread_mutex_lock(&mutex);
```

This kind of deadlock may occur when locking a mutex and then calling a routine that will itself attempt to lock the same mutex. For example:

```
pthread_mutex_t mutex;
struct {
      int a;
      int b;
      int c;
}A;
f()
{
      pthread_mutex_lock(&mutex);   /* call 1 */
      A.a++;
      g();
      A.c =0;
      pthread_mutex_unlock(&mutex);
}
g()
{
      pthread_mutex_lock(&mutex);   /* call 2 */
      A.b += A.a;
      pthread_mutex_unlock(&mutex);
}
```

To avoid this kind of deadlock or data inconsistency, you should use either one of the following locking schemes:

Fine granularity locking Each data atom should be protected by a mutex, locked only by low-level functions. For example, this would result in locking each record of a database. Benefits: High-level functions do not

	need to care about locking data. Drawbacks: It increases the number of mutexes and great care should be taken to avoid deadlocks.
High-level locking	Data should be organized into areas, with each area protected by a mutex; low-level functions do not need to care about locking. For example, this would result in locking a whole database before accessing it. Benefits: There are few mutexes, and thus few risks of deadlocks. Drawbacks: Performance may be degraded, especially if many threads want access to the same data.

Just as thread attributes objects can be used to create threads with non-default attribute values, mutex attributes objects can be used to create mutexes with non-default attribute values. Mutex attributes are part of the POSIX thread options and are not implemented under AIX 5L.

10.4.3 Condition variables

While mutexes can be used to provide a higher level of atomic actions (critical sections), a mechanism for letting a thread go to sleep until a certain condition occurs is often very useful. It could help avoid wasting CPU time when continuously checking if a condition has occurred (polling).

10.4.3.1 Condition variables

A *condition variable* synchronization primitive is provided through POSIX threads. Basically, it permits a thread to *suspend* its execution waiting for a condition or event to be satisfied by the actions of another thread. Once the condition has been met, the thread will be notified and then *resume* execution.

A condition variable is also associated with a shared variable (condition predicate) protected by a mutex. Normally, there will be three variables:

A condition variable	Think of this variable as the part that is used for the suspending and resuming of a thread.
A shared variable	Think of this variable’s value as the information that needs to be shared among the threads. This value is often interpreted as a certain predicate being true (hence the common name condition *predicate*).
A mutex	A mutex is needed to protect the shared variable while it is being manipulated.

The same mutex must be used for the same condition variable, even for different threads. It is possible to bundle the condition variable, the condition (predicate), and the mutex in a structure, as shown in the following code fragment:

```
struct condition_bundle_t {
pthread_cond_t condition_variable;
int condition_predicate;
pthread_mutex_t condition_mutexlock;
};
```

Condition variables can be initialized using condition attributes objects. The attribute which can be set specifies whether or not the condition variable can be used by any thread that has access to the memory where it is allocated, even if these threads belong to different processes. The subroutines are pthread_condattr_setpshared and pthread_condattr_getpshared.

10.4.3.2 Waiting for a condition

When waiting for a condition, the subroutine pthread_cond_wait provided by the POSIX threads atomically unlocks the mutex and blocks the calling thread. When the condition is signaled, the mutex is *relocked*, and the pthread_cond_wait subroutine returns.

It is possible, through pthread_cond_timedwait, to define a period of time that the thread is blocked waiting for the condition, and it can resume either by the condition becoming TRUE or by the expiration of the time-out value.

The pthread_cond_wait and phtread_cond_timedwait subroutines are referred to as condition wait subroutines.

Note

It follows that a thread must always start by locking the mutex before calling a condition wait routine. Also, if a thread holds the mutex, no locked condition wait subroutine calls can return, even the pthread_cond_timedwait, because the semantics of the condition wait subroutines requires that the mutex be relocked at the time of return.

It is important to note that when a condition wait subroutine returns without error, the condition may still be false. The reason is that more than one thread may be awoken. The first thread locking the mutex will block all other awoken threads in the condition wait subroutine until the mutex is unlocked. Thus, the predicate may have changed when the second thread gets the mutex and returns from the condition wait subroutine.

In general, whenever a condition wait returns, the thread should re-evaluate the condition to determine whether it can safely proceed, should wait again, or should declare a timeout. A return from the condition wait subroutine does not imply that the predicate is either true or false.

It is recommended that a condition wait be enclosed in a while-loop that checks the predicate. The following code fragment provides a basic implementation of a condition wait:

```
pthread_mutex_lock(&condition_mutexlock);

while (condition_predicate == 0)
      pthread_cond_wait(&condition_variable, &condition_mutexlock);
...
pthread_mutex_unlock(&condition_mutexlock);
```

In the case where the thread locking the mutex never unlocks it, the threads suspended in a condition wait subroutine can be cancelled (pthread_cancel), if their cancelability is enabled. This is because the condition wait subroutines provide cancellation points (see Table 93 on page 339).

10.4.3.3 Signaling a condition

The pthread_cond_signal subroutine wakes up at least one thread that is currently blocked on the specified condition. The awoken thread is chosen according to the scheduling policy; it is the thread with the most-favored scheduling priority. It may happen on multiprocessor systems, or some non-AIX systems, that more than one thread is woken up. Do not assume that this subroutine wakes up exactly one thread.

The pthread_cond_broadcast subroutine wakes up every thread that is currently blocked on the specified condition. However, a thread can start waiting on the same condition just after the call to the subroutine returns.

10.4.4 Semaphore

The main idea behind semaphores is to control access to a set of resources in the same way a rental car company controls its allocation system. They have a set of cars available and a kind of counter, or semaphore, that shows how many cars are ready in the parking lot. Each time a car is rented, the counter is decremented. Every returned car increases the counter. If a customer requires a car, but the counter is zero, which means no car is available, it must wait until one car becomes available.

This concept is applied on semaphores as a synchronization primitive in traditional UNIX-based interprocess synchronization facilities. If a semaphore is configured to hold the values 0 or 1, it is called a *binary semaphore* and works in the same way as a mutex. But, if it can reach values greater than 1, they can control a set of resources and are called *counting semaphores*.

The decrement and increment operations on semaphores have historically been referred to as P (lock, down) and V (unlock, up).

Semaphores differ from the mutexes and condition variables described in Section 10.4.2, “Mutex” on page 347 and Section 10.4.3, “Condition variables” on page 349 in the following ways:

- A mutex can only be unlocked by the thread which locked it. A semaphore need not be incremented by the same process or thread that decrements it.
- A mutex only has two states: locked or unlocked.

10.4.4.1 An example of interthread semaphores

It is possible to implement interthread semaphores for specific usage. Consider the case where the source system does not support multithreading. Applications using semaphores to control shared data between processes cannot, in general, be meaningfully converted to a single multithreaded process on a system AIX 5L. However, there may be an application which “by nature” could gain from such a conversion. Simulating semaphores using mutexes and condition variables provides a way to test this.

The following implementation is very basic. Error handling is not performed, but cancellations are properly handled with cleanup handlers whenever required.

A semaphore has the sema_t data type. It must be initialized by the sema_init routine and destroyed with the sema_destroy routine. The semaphore request to lock and unlock operations are respectively performed by the sema_p and sema_v routines:

```
typedef struct {
      pthread_mutex_t lock;
      pthread_cond_t cond;
      int count;
} sema_t;
void sema_init(sema_t *sem)
{
      pthread_mutex_init(&sem->lock, NULL);
      pthread_cond_init(&sem->cond, NULL);
      sem->count = 1;
}
```

```
void sema_destroy(sema_t *sem)
{
      pthread_mutex_destroy(&sem->lock);
      pthread_cond_destroy(&sem->cond);
}
void p_operation_cleanup(void *arg)
{
      sema_t *sem;
      sem = (sema_t *)arg;
      pthread_mutex_unlock(&sem->lock);
}
void sema_p(sema_t *sem)
{
      pthread_mutex_lock(&sem->lock);
      pthread_cleanup_push(p_operation_cleanup, sem);
      while (sem->count <= 0)
         pthread_cond_wait(&sem->cond, &sem->lock);
      sem->count--;
      /*
      * Note that the pthread_cleanup_pop subroutine will
      * execute the p_operation_cleanup routine
      */
      pthread_cleanup_pop(1);
}
void sema_v(sema_t *sem)
{
      pthread_mutex_lock(&sem->lock);
      sem->count++;
      if (sem->count <= 0)
         pthread_cond_signal(&sem->cond);
      pthread_mutex_unlock(&sem->lock);
}
```

10.4.5 Porting issues

In AIX 5L, mutexes cannot be relocked by the same thread. This may not be the case on other systems. To enhance portability of your programs, assume that the following code fragment may produce a deadlock:

```
pthread_mutex_lock(&mutex);
pthread_mutex_lock(&mutex);
```

Another point to be aware of when using mutexes to control the access of shared variables is how these variables are aligned in memory and how memory is accessed by the underlying processor architecture. Using mutexes to avoid a situation similar to that described in Section 10.4.1, “Synchronization” on page 345 may not guarantee consistent updating.

If the underlying architecture only accesses memory in units of for example, four bytes, it is possible for the following situation to occur: Several variables, for example, of type char (one byte), are adjacent in memory. Each variable is updated only by a specific thread. However, since the memory access is 4 bytes, and not 1 byte, a thread could effectively overwrite the variable values adjacent to its own variable value in memory (consider again the example in Section 10.4.1, “Synchronization” on page 345).

Clearly, the situation will not change even if the updating of the variables is controlled using a mutex for each variable. The use of mutexes will not guarantee consistent updating of variables, due to the underlying memory access architecture.

The Itanium and Power processors can access the memory in units of 1 byte. As an example, the Alpha processors prior to EV56 (EV4 and EV5) can only access memory in units of at least 4 bytes. This fact may be reflected in the source application’s code (for example, the ordering of a structure’s members) or in the explicit use of compiler options to ensure correct memory alignment.

For the platforms running AIX 5L, the situation described will not occur. There may still be reasons to consider careful memory alignment, such as performance.

Mutex attributes objects cannot be manipulated under AIX 5L. Mutex attributes like protocol, prioceiling, and processshared may be defined on other systems.

10.5 Threads and signals

The signal mechanics is part of UNIX-based systems. It provides a way to handle asynchronous events. Basically, when a process receives a signal, the kernel stops it. Then a piece of code defined as a handler is executed, and after its completion, the process resumes at the exact point where it was before being stopped by the kernel. Each handler is assigned to a specific signal through the signal handler table. Another table called the signal mask defines which signals the process will receive and which it will ignore. Every time a process receives a signal, the kernel checks out its signal mask to determine if it is allowed, and then looks in the signal handler table to execute the proper handler.

10.5.1 Signals

Signal management in multithreaded processes has resulted from a compromise among many and sometimes conflicting goals. The goal of compatibility is assured: signals in multithreaded processes are an extension of signals in traditional single-threaded programs. Programs handling signals and written for single-threaded systems will behave as expected in AIX 5L.

Signal management in multithreaded processes is shared by the process and thread levels, and consists of:

- Per-process signal handlers
- Per-thread signal masks
- Single delivery of each signal

The POSIX threads library also provides a new subroutine and introduces new programming practices for waiting for asynchronously generated signals.

10.5.2 Signal handlers and signal masks

Signal handlers are maintained at process level. It is strongly recommended you only use the sigaction subroutine to get and set signal handlers. Other subroutines may not be supported in the future.

Because the list of signal handlers is maintained at process level, any thread within the process may change it. If two threads set a signal handler on the same signal, the last thread that called the sigaction subroutine will override the setting of the previous thread call; in most cases, it will be impossible to predict the order in which threads are scheduled.

Signal masks are maintained at the thread level. Each thread can have its own set of signals that will be blocked from delivery. The sigthreadmask subroutine must be used to get and set the calling thread's signal mask. The sigprocmask subroutine must not be used in multithreaded programs; otherwise, unexpected behavior may result.

The sigthreadmask subroutine is very similar to sigprocmask. The parameters and usage of both subroutines are exactly the same. When porting existing code to support the threads library, you may simply replace sigprocmask with sigthreadmask.

10.5.3 Signal generation

Signals generated by some action attributable to a particular thread, such as a hardware fault, are sent to the thread that caused the signal to be generated. Signals generated in association with a process ID, a process group ID, or an asynchronous event (such as terminal activity) are sent to the process.

The pthread_kill subroutine sends a signal to a thread. Because thread IDs identify threads within a process, this subroutine can only send signals to threads within the same process. The pthread_kill routine is defined as:

```
int pthread_kill (pthread_t thread, int sig);
```

The kill subroutine (and thus the `kill` command) sends a signal to a process. A thread can send a signal Signal to its process by executing the following call:

```
kill(getpid(), Signal);
```

The raise subroutine cannot be used to send a signal to the calling thread's process. The raise subroutine sends a signal to the calling thread, as in the following call:

```
pthread_kill(pthread_self(), Signal);
```

This ensures that the signal is sent to the caller of the raise subroutine. Thus, library routines written for single-threaded programs may easily be ported to a multithreaded system, because the raise subroutine is usually intended to send the signal to the caller.

The alarm subroutine requests that a signal be sent later to the process, and alarm states are maintained at process level. Thus, the last thread that called the alarm subroutine overrides the settings of other threads in the process. In a multithreaded program, the SIGALRM signal is not necessarily delivered to the thread that called the alarm subroutine. The calling thread may even be terminated; therefore, it cannot receive the signal.

10.5.4 Handling signals

Signal handlers are called within the thread to which the signal is delivered. Signal handlers may call the pthread_self subroutine to get their thread ID. Some limitations to signal handlers are introduced by the threads library:

- Signal handlers may call the longjmp or siglongjmp subroutine only if the corresponding call to the setjmp or sigsetjmp subroutine was performed in the same thread.

  Usually, a program that wants to wait for a signal installs a signal handler that calls the longjmp subroutine to continue execution at the point where the corresponding setjmp subroutine was called. This cannot be done in a multithreaded program, because the signal may be delivered to a thread other than the one that called the setjmp subroutine, thus causing the handler to be executed by the wrong thread.

- Signal handlers must not call the pthread_cond_signal or pthread_cond_broadcast subroutine to signal a condition.

To allow a thread to wait for asynchronously generated signals, the threads library provides the sigwait subroutine.

The sigwait subroutine blocks the calling thread until one of the awaited signals is sent to the process or to the thread. There must not be a signal handler installed for any signal that sigwait is waiting for.

Typically, programs may create a dedicated thread to wait for asynchronously generated signals. Such a thread just loops on a sigwait subroutine call and handles the signals. The following code fragment gives an example of such a signal waiter thread:

```
sigset_t set;
int sig;

sigemptyset(&set);
sigaddset(&set, SIGINT);
sigaddset(&set, SIGQUIT);
sigaddset(&set, SIGTERM);
sigthreadmask(SIG_BLOCK, &set, NULL);

while (1) {
        sigwait(&set, &sig);
        switch (sig) {
                case SIGINT:
                        /* handle interrupts */
                        break;
                case SIGQUIT:
                        /* handle quit */
                        break;
                case SIGTERM:
                        /* handle termination */
                        break;
                default:
                        /* unexpected signal */
                        pthread_exit((void *)-1);
        }
}
```

If more than one thread called the sigwait subroutine, exactly one call returns when a matching signal is sent. There is no way to predict which thread will be awakened. Note that the sigwait subroutine provides a cancellation point.

Because a dedicated thread is not a real signal handler, it may signal a condition to any other thread. It is possible to implement a sigwait_multiple routine that would awaken all threads waiting for a specific signal. Each caller of the sigwait_multiple routine would register a set of signals. The caller then waits on a condition variable. A single thread calls the sigwait subroutine on the union of all registered signals. When the call to the sigwait subroutine returns, the appropriate state is set and condition variables are broadcasted. New callers to the sigwait_multiple subroutine would cause the pending sigwait subroutine call to be canceled and reissued to update the set of signals being waited for.

10.5.5 Signal delivery

A signal is delivered to a thread, unless its action is set to ignore. The following rules govern signal delivery in a multithreaded process:

- A signal whose action is set to terminate, stop, or continue the target thread or process, respectively, terminates, stops, or continues the entire process (and thus all of its threads). This means that single-threaded programs may be rewritten as multithreaded programs without changing their externally visible signal behavior.

  Consider, for example, a multithreaded user command, such as the `grep` command. A user may start the command in his favorite shell and then decide to stop it by sending a signal with the `kill` command. It is obvious that the signal should stop the entire process running the `grep` command.
- Signals generated for a specific thread, using the pthread_kill or the raise subroutines, are delivered to that thread. If the thread has blocked the signal from delivery, the signal is set pending on the thread until the signal is unblocked from delivery. If the thread is terminated before signal delivery, the signal will be ignored.
- Signals generated for a process, using the kill subroutine, for example, are delivered to exactly one thread in the process. If one or more threads called the sigwait subroutine, the signal is delivered to exactly one of these threads. Otherwise, the signal is delivered to exactly one thread that did not block the signal from delivery. If no thread matches these conditions, the signal is set pending on the process until a thread calls the sigwait subroutine specifying this signal or a thread unblocks the signal from delivery.

If the action associated with a pending signal (on a thread or on a process) is set to ignore, the signal is ignored.

10.5.6 Porting issues

AIX 5L supports the signals defined in Section 7.6, “Signals” on page 194. The set of supported signals can be found in the systems signal.h header file. Note that this set may very well be different on the source system.

10.6 Thread specific data

When converting non-threaded applications to threaded, data access will now possibly be attempted by several threads. As all threads share the same process address space, they also share the same data space.

Thread-specific data (TSD) is a POSIX functionality that permits creation of per-thread data. This allows multiple threads to run the same code and access thread-specific data using the same variable names. This makes the design of the code easier, because it does not need to be aware of which thread is running and is a common programming technique when converting functions into thread versions.

10.6.1 Keys

Thread-specific data may be viewed as a two-dimensional array of values, with keys serving as the row index and thread IDs as the column index, as shown in Figure 66 on page 360.

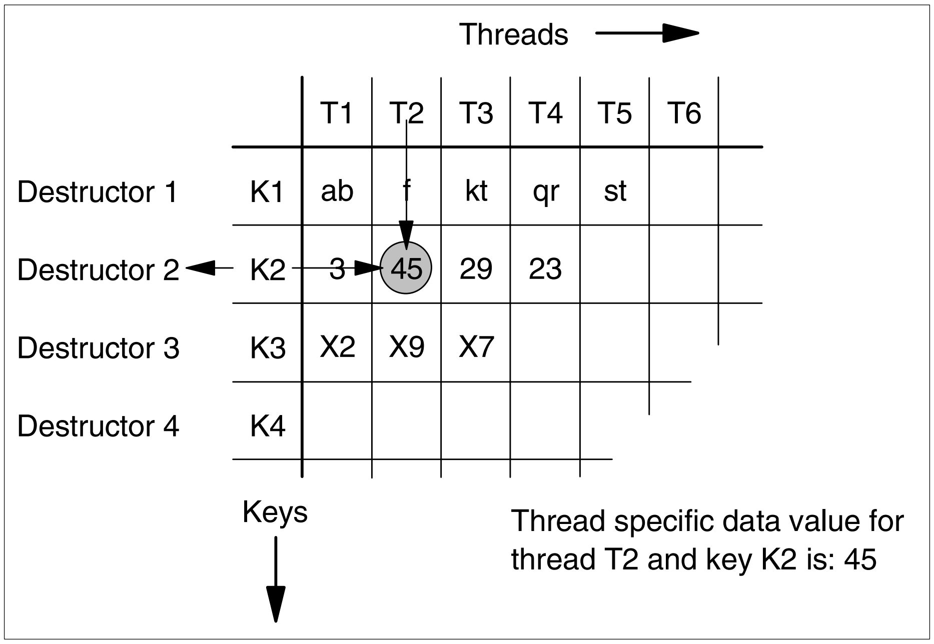


Figure 66. Thread specific data, simplified view

A thread-specific data key is an opaque object of type pthread_key_t. The same key can be used by all threads in a process. Although all threads use the same key, they set and access different thread-specific data values associated with that key. Thread-specific data are void pointers. This allows referencing any kind of data, such as dynamically allocated strings or structures.

The pthread_key_create subroutine creates a thread-specific data key:

```
int pthread_key_create ( pthread_key_t * key, void (*destructor) (void *));
```

The key is shared among all threads within the process, but each thread has its own data associated with the key. The thread-specific data is initially set to NULL.

The application is responsible for ensuring that this subroutine is called only once for each requested key. This can be done, for example, by calling the subroutine before creating other threads or by using the one-time initialization facility.

At the key creation time, an optional destructor routine can be specified. If the key specific value is not NULL, that destructor will be called for each thread terminated and detached. Typically, the destructor routine will release the storage thread-specific data. It will receive the thread-specific data as a parameter.

To summarize, for each key there is associated (at most) one destructor routine, which will be called for all threads, but an individual value for each pair of key and thread. See Figure 66 on page 360 for reference.

For example, a thread-specific data key may be used for dynamically allocated buffers. A destructor routine should be provided to ensure that the buffer is freed when the thread terminates. The free subroutine can be used:

```
pthread_key_create(&key, free);
```

More complex destructors may be used as shown in the following:

```
typedef struct {
      FILE *stream;
      char *buffer;
} data_t;
...
void destructor(void *data)
{
      fclose(((data_t *)data)->stream);
      free(((data_t *)data)->buffer);
      free(data);
      *data = NULL;
}
```

Thread-specific data is accessed using the pthread_getspecific and pthread_setspecific subroutines.

```
void *pthread_getspecific (pthread_key_t key);
void *pthread_setspecific (pthread_key_t key, const void *value);
```

The first one reads the value bound to the specified key and specific to the calling thread; the second one sets the value, as shown in the following code example:

```
pthread_key_create(&key, free);
...
private_data = malloc(...);
pthread_setspecific(key, private_data);
...
pthread_getspecific(key, &data);
...
```

10.6.2 Porting issues

Although some implementations of the threads library may repeat destructor calls, the destructor routine is called only once in AIX 5L. Take care when porting code from other systems where a destructor routine can be called several times.

It is possible to store values that are not pointers, such as integers. It is not recommended to do this for at least two reasons:

- Casting a pointer into a scalar type may not be portable.
- The `NULL` pointer value is implementation-dependent; several systems assign the `NULL` pointer a non-zero value.

If you are sure that your program will never be ported to another system, you may use integer values for thread-specific data.

10.7 Compiling and linking

When compiling and linking threaded applications, it is important to use the correct libraries. The notions of reentrant functions and thread safe functions will be explained with an overview of compiler invocations.

10.7.1 Reentrant functions and thread safe functions

One very important point to take care of when building multithreaded programs is the resource handling. To avoid getting in trouble, be sure to create only thread-safe and reentrant functions as much as possible. Re-entrance and thread-safety are separate concepts; a function can be either reentrant, thread-safe, both, or neither.

Reentrant	A reentrant function does not hold static data over successive calls, nor does it return a pointer to static data. All data is provided by the caller of the function. A reentrant function must not call non-reentrant functions.
Thread-safe	A thread-safe function protects shared resources from concurrent access by locks. Thread-safety concerns only the implementation of a function and does not affect its external interface. The use of global data is thread-unsafe. It should be maintained per thread or encapsulated so that its access can be serialized.

Reentrant and thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming environments, not just within threads. Thus, it is good programming practice to always use and write reentrant and thread-safe functions.

Several libraries shipped with the AIX Base Operating System are thread-safe. In AIX 5L, the libraries are thread-safe, except for the routines explicitly listed in Table 96.

Table 96. List of AIX interfaces that are not thread-safe.

Library name	**Not thread-safe**
libc.a	Standard functions: advance, asctime, brk, catgets, chroot, compile, ctime, cuserid, dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store, dirname, drand48, ecvt, encrypt, endgrent, endpwent, endutxent, fcvt, gamma, gcvt, getc_unlocked, getchar_unlocked, getdate, getdtablesize, getenv, getgrent getgrgid, getgrnam, getlogin, getopt, getpagesize, getpass, getpwent, getpwnam, getpwuid, getutxent, getutxid getutxline, getw, getw, gmtime, l64a, lgamma, localtime, lrand48, mrand48, nl_langinfo, ptsname, putc_unlocked, putchar_unlocked, putenv, pututxline, putw, rand, random, readdir, re_comp, re_exec, regcmp, regex, sbrk, setgrent, setkey, setlocale, setpwent, setutxent, sigstack, srand48, srandom, step, strerror, strtok, ttyname, ttyslot, wait3 AIX specific functions: endfsent, endttyent, endutent, getfsent, getfsfile, getfsspec, getfstype, getttyent, getttynam, getutent, getutid, getutline, pututline, setfsent, setttyent, setutent, utmpname
libbsd.a	timezone
libm.a and libmsaa.a	gamma, lgamma
libPW.a, libblas.a, libcur.a, libcurses.a, libplot.a, libprint.a	All functions

Some of the standard C subroutines are non-reentrant, such as the ctime and strtok subroutines. The reentrant version of the subroutines have the name of the original subroutine with a suffix _r (underscore r).

10.7.2 Compiling and linking

The POSIX standard for the threads library specifies the implementation of some parts as optional. All subroutines defined by the threads library API are always available. Depending on the available options, some subroutines may not be implemented.

Unimplemented subroutines can be called by applications, but they always return the ENOSYS error code.

Symbolic constants (symbols) can be used to get the availability of options on the system where the program is compiled. The symbols are defined in the pthread.h header file by the `#define` pre-processor command. For unimplemented options, the corresponding symbol is undefined by the `#undef` pre-processor command. Checking option symbols should be done in each program that may be ported to another system.

The following list indicates the symbol associated with each option (see also Section 10.10.4, "POSIX options" on page 378):

Stack address	_POSIX_THREAD_ATTR_STACKADDR
Stack size	_POSIX_THREAD_ATTR_STACKSIZE
Priority scheduling	_POSIX_THREAD_PRIORITY_SCHEDULING
Priority inheritance	_POSIX_THREAD_PRIO_INHERIT
Priority protection	_POSIX_THREAD_PRIO_PROTECT
Process sharing	_POSIX_THREAD_PROCESS_SHARED

The simplest action to take when an option is not available is to stop the compilation, as in the following example:

```
#ifndef _POSIX_THREAD_ATTR_STACKSIZE
#error "The stack size POSIX option is required"
#endif
```

The pthread.h header file also defines the following symbols that can be used by other header files or by programs:

_POSIX_REENTRANT_FUNCTIONS	Denotes that reentrant functions are required.
_POSIX_THREADS	Denotes the implementation of the threads library.

It is also possible to use the sysconf routine to get the availability of options on the system where the program is executed. The symbolic constant passed as the Name parameter for the sysconf routine is obtained by substituting _SC for _POSIX in the symbolic constants listed above. For example, _POSIX_THREAD_PRIO_INHERIT will become _SC_THREAD_PRIO_INHERIT.

In AIX 5L, compiling and linking a multithreaded application is as simple as compiling a non-threaded application.

Table 97 shows all important information about the compiler mode, specifically the compiler driver program to use, depending on the required pthreads standard.

Table 97. AIX 5L C driver programs

C driver program	Description
xlc_r cc_r xlC_r	All _r-suffixed invocations are functionally similar to their corresponding base compiler invocation, but set the macro name -D_THREAD_SAFE and invoke the added compiler options: • L/usr/lib/threads • L/usr/lib/dce • lpthreads • qthreaded Use the _r-suffixed invocations when compiling with the -qsmp compiler option or if you want to create POSIX threaded applications.
xlc_r4 cc_r4 xlC_r4	Use _r4-suffixed invocations to provide compatibility between DCE applications written for AIX Version 3.2.5 and AIX Version 4. They link your application to the correct AIX Version 4 DCE libraries, providing compatibility between the latest version of the pthreads library and the earlier versions supported on AIX Version 3.2.5.
xlc_r7 cc_r7 xlC_r7	Use the _r7-suffixed invocations to compile and link applications conforming to the POSIX Draft 7 standard. Otherwise, the compiler will, by default, compile and link applications conforming to the current POSIX threads standards.

The xlc driver compiles C source code with a default language level as ANSI, and cc compiles C sources with default language level as extended. The extended level is suitable for code that does not require full compliance with the ANSI C standard, for example, legacy code. The xlC driver is for C++ code.

In the following example, a very simple makefile is suggested to compile and link a multithreaded C program called ex1.c:

```
CC = /usr/vac/bin/xlc_r
CFLAGS = -g
BIN = ex1
```

```
all: $(BIN)
clean:
rm -f $(BIN)
rm -f *.o
```

Notice that the default path for the C compiler is /usr/vac/bin.

10.7.3 Porting issues

When writing multithreaded programs, the reentrant versions of subroutines should be used instead of the original version.

When including the header file pthread.h, this file must be the first included (see the example in Section 10.1.1.3, “Example of a POSIX threaded program” on page 310).

For multithreaded applications, you must use one of the _r-suffixed C driver programs.

The C for AIX compiler offers you two methods of implementing shared memory program parallelization. These are:

- Automatic and explicit parallelization of countable loops using IBM `pragma` directives.
- Program parallelization using `pragma` directives compliant to the OpenMP Application Program Interface specification.

All methods of program parallelization are enabled when the -qsmp compiler option is in effect without the omp suboption. You can enable strict OpenMP compliance with the -qsmp=omp compiler option, but doing so will disable automatic parallelization.

Parallel regions of program code are executed by multiple threads, possibly running on multiple processors. The number of threads created is determined by run-time options and calls to library functions.

Work is distributed among available threads according to the directives specified in the source.

When programming C++, you have the choice of using the IBM Open Class. The classes support multithreading and all functions are thread-safe, unless documented otherwise. For more information, see:

`http://www.ibm.com/software/ad/vacpp/library.html`

10.8 Tuning

AIX Version 4.3.1 replaced the previous 1:1 threads implementation model with an M:N version. The M:N model complies with the UNIX 98 pthreads standard, which includes the POSIX threads standard. Previous releases of AIX Version 4 complied with Draft 7 of the POSIX pthreads standard. AIX 5L is binary compatible with previous releases. The UNIX 98 implementation is the default for application development, but you can use specific compiler drivers, as shown in Table 97 on page 365, to develop new applications using Draft 7 pthreads.

The M:N pthreads implementation provides several environment variables that can be used to affect application performance. If possible, the application developer should provide a front-end shell script to invoke the binary executable in which the user may specify new values to override the system defaults. The following environment variables can be set by end users and are examined at process initialization time:

AIXTHREAD_SCOPE
This variable can be used to set the contention scope of pthreads created using the default pthread attribute object. It is represented by the following syntax:

AIXTHREAD_SCOPE=[P|S]

The value P indicates process scope, while a value of S indicates system scope. If no value is specified, the default pthread attribute object will use process scope contention, which implies the M:N model.

SPINLOOPTIME
This variable controls the number of times the system will try to get a busy lock without taking a secondary action, such as calling the kernel to yield the processor. This control is really intended for SMP systems, where it is hoped that the lock is held by another actively running pthread and will soon be released. On uniprocessor systems, this value is ignored.

YIELDLOOPTIME
This variable controls the number of times that the system yields the processor when trying to acquire a busy mutex or spin lock before actually going to sleep on the lock. This variable has been shown to be effective

in complex applications where multiple locks are in use.

The following environment variables impact the scheduling of pthreads created with process-based contention scope:

AIXTHREAD_MNRATIO This variable allows the user to specify the ratio of pthreads to kernel threads. It is examined when creating a pthread to determine if a kernel thread should also be created to maintain the correct ratio. It is represented with the following syntax:

AIXTHREAD_MNRATIO=p:k

where k is the number of kernel threads to use to handle p pthreads. Any positive integer value may be specified for p and k.

These values are used in a formula that employs integer arithmetic, which can result in the loss of some precision when big numbers are specified. If k is greater than p, the ratio is treated as 1:1. If no value is specified, the default ratio depends on the default contention scope. If system scope contention is the default, the ratio is 1:1. If process scope contention is set as the default, the ratio is 8:1.

Note

When migrating threaded applications to AIX from other platforms or previous versions of AIX, the default 8:1 ratio used with the M:N threads model may reduce application performance.

If this is the case, you can either change the source code of the application so that threads are created with the contention scope attribute set to PTHREAD_SCOPE_SYSTEM, set the AIXTHREAD_SCOPE environment variable to the value S, or change the ratio of kernel threads to user threads with the AIXTHREAD_MNRATIO environment variable.

AIXTHREAD_SLPRATIO This variable is used to determine the number of kernel threads used to support local pthreads sleeping in the library code on a pthread event, for example, attempting to obtain a mutex. It is represented by the following syntax:

AIXTHREAD_SLPRATIO=k:p

where k is the number of kernel threads to reserve for every p sleeping pthreads. Notice that the relative positions of the numbers indicating kernel threads and user pthreads are reversed when compared with AIXTHREAD_MNRATIO. Any positive integer value may be specified for p and k. These values are used in a formula that employs integer arithmetic, which can result in the loss of some precision when large numbers are specified. If k is greater than p, the ratio is treated as 1:1. If the variable is not set, a ratio of 1:12 is used. The reason for maintaining kernel threads for sleeping pthreads is that, when the pthread event occurs, the pthread will immediately require a kernel thread to run on. It is more efficient to use a kernel thread that is already available than it is to create a new kernel thread once the event has taken place.

AIXTHREAD_MINKTHREADS This variable is a manual override to the AIXTHREAD_MNRATIO. It allows you to stipulate the minimum number of active kernel threads. The library scheduler will not reclaim kernel threads below this number.

10.9 Multiheap malloc

By default, the malloc subsystem uses a single heap or free memory pool. Starting with AIX Version 4.3.3, the malloc routine supports an optional multiheap capability to allow applications to enable the use of multiple heaps of free memory, rather than just one.

The purpose of providing multiple heap capability in the malloc subsystem is to improve the performance of threaded applications running on multiprocessor systems. When the malloc subsystem is limited to using a single heap, simultaneous memory allocation requests received from threads running on separate processors are serialized, meaning that the malloc subsystem can only service one thread at a time. This can have a serious impact on application performance.

With the multiheap capability enabled, the malloc subsystem creates a fixed number of heaps for its use. Each memory allocation request is serviced using one of the available heaps. The malloc subsystem can then process memory allocation requests in parallel as long as the number of threads simultaneously requesting service is less than or equal to the number of heaps.

10.9.1 Using multiheap malloc

The simplest way of using the malloc multiheap feature is to set the following environment variable:

```
MALLOCMULTIHEAP=true
```

This enables the feature with the default configuration of 32 memory pools.

Multithreaded C++ programs will potentially also have a large benefit from using the malloc multiheap feature, because each heap must be accessed each time a constructor or destructor is called.

10.9.2 Parameters of malloc multiheap

The malloc multiheap feature also offers tuning parameters to alter the number of heaps from the default of 32 and alter the algorithm to select the heap to be used.

10.9.2.1 The number of heaps

If it is enabled, the malloc multiheap feature uses 32 heaps by default. If you know you will not use as many processors, or for any other reason, you can ask for any lower number of heaps. Instead of setting the environment variable MALLOCMULTIHEAP to a value of true, it is set to a value of heaps: n, where n is the number of heaps that are desired.

Heaps are allocated in a round-robin way. This means all heaps are used, whether they are needed or not. A more subtle (though more time-consuming) way to allocate space would be to use the first available heap instead of the next one. The considersize option allows this.

10.9.2.2 The considersize option

By default, malloc multiheap selects a new, available heap every time a request is made, essentially using round-robin selection. The considersize option will select, instead, the first available heap that has enough free space to handle the request. While somewhat slower in computation time, this option can help reduce both the working set size and the number of sbrk() calls. The considersize option is specified when setting the MALLOCMULTIHEAP environment variable, along with the number of required heaps as follows:

```
MALLOCMULTIHEAP=heaps:4,considersize
```

10.10 Quick reference

This section is intended as a quick reference to the POSIX threads implementation under AIX 5L.

10.10.1 AIX implementations of threads

The AIX files shown in Table 98 provide the AIX implementation of pthreads.

Table 98. AIX implementation of threads

File	Description
/usr/include/pthread.h	Header file with most pthread definitions
/usr/include/sched.h	Header file with some scheduling definitions
/usr/include/unistd.h	Header file with pthread_atfork() definition
/usr/include/sys/limits.h	Header file with some pthread definitions
/usr/include/sys/pthdebug.h	Header file with most pthread debug definitions
/usr/include/sys/sched.h	Header file with some scheduling definitions
/usr/include/sys/signal.h	Header file with pthread_kill() and pthread_sigmask() definitions
/usr/include/sys/types.h	Header file with some pthread definitions
/usr/lib/libpthreads.a	32-bit/64-bit library providing UNIX 98 and POSIX 1003.1c pthreads (Power)
/usr/lib/libpthreads_compat.a	32-bit only library providing POSIX 1003.1c Draft 7 pthreads
/usr/lib/profiled/libpthreads.a	Profiled 32-bit/64-bit library providing UNIX 98 and POSIX 1003.1c pthreads
/usr/lib/profiled/libpthreads_compat.a	Profiled 32-bit only library providing POSIX 1003.1c 7 pthreads

10.10.2 POSIX interfaces

Table 99 shows which POSIX thread routines are implemented for various major platforms. A yes in the column means that the routine is implemented and not merely declared to return ENOSYS.

Table 99. POSIX threads

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_atfork (void (*prepare)(void), void (*parent)(void) void (*child)(void))`	yes	yes	yes	yes
`int pthread_attr_destroy (pthread_attr_t *attr)`	yes	yes	yes	yes
`int pthread_attr_getdetachstate (const pthread_attr_t *attr, int * detachstate)`	yes	yes	yes	yes
`int pthread_attr_getinheritsched (const pthread_attr_t *attr, int *inheritsched)`	yes	yes	yes	yes
`int pthread_attr_getschedparam (const pthread_attr_t *attr, struct sched_param *schedparam)`	yes	yes	yes	yes
`int pthread_attr_getschedpolicy (const pthread_attr_t *attr, int *policy)`	yes	yes	yes	yes
`int pthread_attr_getscope (const pthread_attr_t *attr, int *contentionscope)`	yes	yes	yes	yes
`int pthread_attr_getstackaddr (const pthread_attr_t *attr, void **stackaddr)`	yes	yes	yes	yes
`int pthread_attr_getstacksize (const pthread_attr_t *attr, size_t *stacksize)`	yes	yes	yes	yes
`int pthread_attr_init (pthread_attr_t *attr)`	yes	yes	yes	yes
`int pthread_attr_setdetachstate (pthread_attr_t *attr, int detachstate)`	yes	yes	yes	yes

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_attr_setinheritsched (pthread_attr_t *, int inhertisched)`	yes	yes	yes	yes
`int pthread_attr_setschedparam (pthread_attr_t *attr, const struct sched_param *schedparam)`	yes	yes	yes	yes
`int pthread_attr_setschedpolicy (pthread_attr_t *attr, int policy)`	yes	yes	yes	yes
`int pthread_attr_setscope (pthread_attr_t *attr, int contentionscope)`	yes	yes	yes	yes
`int pthread_attr_setstackaddr (pthread_attr_t *attr, void *stackaddr)`	yes	yes	yes	yes
`int pthread_attr_setstacksize (pthread_attr_t *attr, size_t stacksize)`	yes	yes	yes	yes
`int pthread_cancel (pthread_t thread)`	yes	yes	yes	yes
`void pthread_cleanup_pop (int execute)`	yes	yes	yes	yes
`void pthread_cleanup_push (void (*routine)(void *), void *arg)`	yes	yes	yes	yes
`int pthread_cond_broadcast (pthread_cond_t *condition)`	yes	yes	yes	yes
`int pthread_cond_destroy (pthread_cond_t *cond)`	yes	yes	yes	yes
`int pthread_cond_init (pthread_cond_t *cond, const pthread_condattr_t *attr)`	yes	yes	yes	yes
`int pthread_cond_signal (pthread_cond_t *condition)`	yes	yes	yes	yes
`int pthread_cond_timedwait (pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime)`	yes	yes	yes	yes

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_cond_wait (pthread_cond_t *, pthread_mutex_t *)`	yes	yes	yes	yes
`int pthread_condattr_destroy (pthread_condattr_t *attr)`	yes	yes	yes	yes
`int pthread_condattr_getpshared (const pthread_condattr_t *attr, int *pshared)`	yes	yes	yes	yes
`int pthread_condattr_init (pthread_condattr_t *attr)`	yes	yes	yes	yes
`int pthread_condattr_setpshared (pthread_condattr_t *attr, int pshared)`	yes	yes	yes	yes
`int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void), void *arg)`	yes	yes	yes	yes
`int pthread_detach (pthread_t thread)`	yes	yes	yes	yes
`int pthread_equal (pthread_t t1, pthread_t t2)`	yes	yes	yes	yes
`void pthread_exit (void *value_ptr)`	yes	yes	yes	yes
`int pthread_getschedparam (pthread_t thread, int *schedpolicy, struct sched_param *schedparam)`	yes	yes	yes	yes
`void *pthread_getspecific (pthread_key_t key)`	yes	yes	yes	yes
`int pthread_join (pthread_t thread, void **value_ptr)`	yes	yes	yes	yes
`int pthread_key_create (pthread_key_t * key, void (*destructor) (void *))`	yes	yes	yes	yes
`int pthread_key_delete (pthread_key_t key)`	yes	yes	yes	yes

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_kill (pthread_t thread, int sig)`	yes	yes	yes	yes
`int pthread_mutex_destroy (pthread_mutex_t *mutex)`	yes	yes	yes	yes
`int pthread_mutex_getprioceiling (const pthread_mutexattr_t *attr, int *prioceiling)`	no	yes	no	yes
`int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)`	yes	yes	yes	yes
`int pthread_mutex_lock (pthread_mutex_t *mutex)`	yes	yes	yes	yes
`int pthread_mutex_setprioceiling (pthread_mutexattr_t *attr, int prioceiling int *oldceiling)`	no	yes	no	yes
`int pthread_mutex_trylock (pthread_mutex_t *mutex)`	yes	yes	yes	yes
`int pthread_mutex_unlock (pthread_mutex_t *mutex)`	yes	yes	yes	yes
`int pthread_mutexattr_destroy (pthread_mutexattr_t *attr)`	yes	yes	yes	yes
`int pthread_mutexattr_getprioceiling (const pthread_mutexattr_t *attr, int *prioceiling)`	no	yes	no	yes
`int pthread_mutexattr_getprotocol (const pthread_mutexattr_t *attr, int *protocol)`	no	yes	no	yes
`int pthread_mutexattr_getpshared (const pthread_mutexattr_t *attr, int *pshared)`	yes	yes	yes	yes
`int pthread_mutexattr_init (pthread_mutexattr_t *attr)`	yes	yes	yes	yes
`int pthread_mutexattr_setprioceiling (pthread_mutexattr_t *attr, int prioceiling int *oldceiling)`	no	yes	no	yes

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr, int protocol)`	no	yes	no	yes
`int pthread_mutexattr_setpshared (pthread_mutexattr_t *attr, int pshared)`	yes	yes	yes	yes
`int pthread_once (pthread_once_t *once_control, void (*init_routine)(void))`	yes	yes	yes	yes
`pthread_t pthread_self (void)`	yes	yes	yes	yes
`int pthread_setcancelstate (int state, int *oldstate)`	yes	yes	yes	yes
`int pthread_setcanceltype (int type, int *oldstype)`	yes	yes	yes	yes
`int pthread_setschedparam (pthread_t thread, int schedpolicy, const struct sched_param *schedparam)`	yes	yes	yes	yes
`int pthread_setspecific (pthread_key_t key, const void *value)`	yes	yes	yes	yes
`int pthread_sigmask (int how, const sigset_t *set, sigset_t *oset)`	yes	yes	yes	yes
`void pthread_testcancel (void)`	yes	yes	yes	yes

10.10.3 X/Open UNIX 98 thread interfaces

Table 100 lists the X/Open UNIX 98 thread routines implemented for AIX 5L and other major implementations.

Table 100. X/Open UNIX 98

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_attr_getguardsize (const pthread_attr_t *attr, size_t *guardsize)`	yes	yes	yes	yes

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_attr_setguardsize (pthread_attr_t *attr, size_t guardsize)`	yes	yes	yes	yes
`int pthread_continue (pthread_t thread)`	yes	no	no	yes
`int pthread_getconcurrency (void)`	yes	yes	yes	yes
`int pthread_mutexattr_gettype (pthread_mutexattr_t *attr, int *type)`	yes	yes	yes	yes
`int pthread_mutexattr_settype (pthread_mutexattr_t *attr, int type)`	yes	yes	yes	yes
`int pthread_rwlock_destroy (pthread_rwlock_t *rwlock)`	yes	yes	yes	yes
`int pthread_rwlock_init (pthread_rwlock_t *rwlock, const pthread_rwlock attr_t *attr)`	yes	yes	yes	yes
`int pthread_rwlock_rdlock ( pthread_rwlock_t *rwlock)`	yes	yes	yes	yes
`int pthread_rwlock_tryrdlock (pthread_rwlock_t *rwlock)`	yes	yes	yes	yes
`int pthread_rwlock_trywrlock (pthread_rwlock_t *rwlock)`	yes	yes	yes	yes
`int pthread_rwlock_unlock (pthread_rwlock_t *rwlock)`	yes	yes	yes	yes
`int pthread_rwlock_wrlock (pthread_rwlock_t *rwlock)`	yes	yes	yes	yes
`int pthread_rwlockattr_destroy (pthread_rwlockattr_t *attr)`	yes	yes	yes	yes
`int pthread_rwlockattr_getpshared (const pthread_rwlockattr_t *attr, int *pshared)`	yes	yes	yes	yes
`int pthread_rwlockattr_init (pthread_rwlockattr_t *attr)`	yes	yes	yes	yes

Routine	AIX 5L	Solaris 8	Tru64	HP-UX 11i
`int pthread_rwlockattr_setpshared (pthread_rwlockattr_t *attr, int pshared)`	yes	yes	yes	yes
`int pthread_setconcurrency (int new_level)`	yes	yes	yes	yes
`int pthread_suspend (pthread_t thread)`	yes	no	no	yes

10.10.4 POSIX options

Table 101 lists the POSIX thread options supported by AIX 5L. In Section 10.10.7, “Limits and default values” on page 386, a program is listed, which extracts relevant values for a given system.

Table 101. Supported POSIX thread options for AIX 5L

AIX 5L	Description
_POSIX_THREAD_ATTR_ STACKADDR	stackaddr attribute for threads is supported.
_POSIX_THREAD_ATTR_ STACKSIZE	stacksize attribute for threads is supported.
_POSIX_THREAD_PROCESS_ SHARED	Cross-process synchronization is supported.
_POSIX_THREAD_SAFE_ FUNCTIONS	Thread-safe libraries are supported.
_POSIX_THREADS	Threads are supported.

POSIX options not supported are shown in Table 102.

Table 102. Not supported POSIX thread options for AIX 5L

AIX 5L	Description
_POSIX_THREAD_PRIORITY_ SCHEDULING	Priority scheduling for threads is supported.
_POSIX_THREAD_PRIO_INHERIT	Priority inheritance supported.
_POSIX_THREAD_PRIO_PROTECT	Priority protection supported.

10.10.5 Supported thread models

Table 103 shows which thread models are supported for major platforms.

Table 103. Supported thread models

Operating System	Thread model
AIX 3.2	M:1
AIX 4.1 and 4.2	1:1
AIX 4.3 and AIX 5L	M:N
Solaris 7 and 8	M:N
HP-UX 11.0 and 11.i	1:1
Tru64	M:N

10.10.6 Mappings to POSIX/UNIX 98 threads

The following tables contain mappings of common thread libraries to POSIX/UNIX 98 threads. The mappings are not semantically preserving, but give an overview of the similarities, not eliminating the need to consult the manual page for each of the routines for complete details. For example, it may be the case that routines have the same names, but different prototypes.

The main differences, apart from the obvious syntactical changes include:

- POSIX threads characteristics are set using configurable attribute objects for each thread.
- POSIX thread routines now return error codes and do not set the global errno.
- POSIX thread routines enhance thread scheduling and enforce scheduling algorithms.
- POSIX specification of cancellation points.
- POSIX threads implement thread cancellation.
- POSIX threads does not support suspending and resuming.
- POSIX threads signal handling.
- POSIX thread routines enhance thread-specific data handling.
- POSIX threads allow for clean-up handlers for fork calls.

AIX 5L does have _np routines; see Table 91 on page 322, which could have been listed in the mapping tables, where the entry is None. However, for the sake of portability, the use of the _np routines is not encouraged. Also, non-pthread routines could also have been used in the mapping. For example, pthread_delay_np could be mapped to nanosleep.

The mapping of the Solaris threads to the POSIX/UNIX 98 threads is shown in Table 104.

Table 104. Mapping of Solaris threads to POSIX/UNIX 98 threads

Solaris threads	**POSIX/UNIX 98**
cond_broadcast	pthread_cond_broadcast
cond_destroy	pthread_cond_destroy
cond_init	pthread_cond_init
cond_signal	pthread_cond_signal
cond_timedwait	pthread_cond_timedwait
cond_wait	pthread_cond_wait
mutex_destroy	pthread_destroy
mutex_init	pthread_mutex_init
mutex_lock	pthread_mutex_lock
mutex_trylock	pthread_mutex_trylock
mutex_unlock	pthread_unlock
rw_rdlock	pthread_rwlock_rdlock
rw_tryrdlock	pthread_rwlock_tryrdlock
rw_trywrlock	pthread_rwlock_trywrlock
rw_unlock	pthread_rwlock_unlock
rw_wrlock	pthread_rwlock_wrlock
rwlock_destroy	pthread_rwlock_destroy
rwlock_init	pthread_rwlock_init
thr_continue	None
thr_create	pthread_create
thr_exit	pthread_exit
thr_getconcurrency	pthread_getconcurrency
thr_getprio	pthread_getschedparam

Solaris threads	POSIX/UNIX 98
thr_getspecific	pthread_getspecific
thr_join	pthread_join
thr_keycreate	pthread_key_create
thr_kill	pthread_kill
thr_main	None
thr_min_stack	None
thr_self	pthread_self
thr_setconcurrency	pthread_setconcurrency
thr_setprio	pthread_setschedparam
thr_setsigmask	pthread_sigmask
thr_setspecific	pthread_setspecific
thr_suspend	None
thr_yield	None

The CMA (Concert Multithread Architecture) threads are based on the POSIX Draft 4. CMA threads are also referred to as DCE threads or user-space threads. The mapping of the CMA threads to the POSIX/UNIX threads is shown in Table 105.

Table 105. Mapping of Compaq Tru64 CMA threads to POSIX/UNIX 98 threads

Tru64 CMA interface	POSIX/UNIX 98
cma_alert_disable_asynch	pthread_setcancelstate pthread_setcanceltype
cma_alert_disable_general	pthread_setcancelstate pthread_setcanceltype
cma_alert_enable_asynch	pthread_setcancelstate pthread_setcanceltype
cma_alert_enable_general	pthread_setcancelstate pthread_setcanceltype
cma_alert_restore	pthread_setcancelstate pthread_setcanceltype
cma_alert_test	pthread_testcancel

Tru64 CMA interface	POSIX/UNIX 98
cma_attr_create	pthread_attr_init
cma_attr_delete	pthread_attr_destroy
cma_attr_get_guardsize	pthread_attr_getguardsize
cma_attr_get_inherit_sched	pthread_attr_getinheritsched
cma_attr_get_mutex_kind	pthread_mutexattr_gettype
cma_attr_get_priority	pthread_attr_getschedparam
cma_attr_get_sched	pthread_attr_getschedpolicy
cma_attr_get_stacksize	pthread_attr_getstacksize
cma_attr_set_guardsize	pthread_attr_setguardsize
cma_attr_set_inherit_sched	pthread_attr_setinheritsched
cma_attr_set_mutex_kind	pthread_mutexattr_settype
cma_attr_set_priority	pthread_attr_setschedparam
cma_attr_set_sched	pthread_attr_setschedpolicy
cma_attr_set_stacksize	pthread_attr_setstacksize
cma_cond_broadcast	pthread_cond_broadcast
cma_cond_create	pthread_cond_init
cma_cond_delete	pthread_cond_destroy
cma_cond_signal	pthread_cond_signal
cma_cond_signal_int	None
cma_cond_timed_wait	pthread_cond_timedwait
cma_cond_wait	pthread_cond_wait
cma_delay	None
cma_handle_assign	None
cma_handle_equal	pthread_equal
cma_init	None
cma_key_create	pthread_key_create
cma_key_get_context	pthread_getspecific

Tru64 CMA interface	POSIX/UNIX 98
cma_key_set_context	pthread_setspecific
cma_lock_global	None
cma_mutex_create	pthread_mutex_init
cma_mutex_delete	pthread_mutex_destroy
cma_mutex_lock	pthread_mutex_lock
cma_mutex_try_lock	pthread_mutex_trylock
cma_mutex_unlock	pthread_mutex_unlock
cma_once	pthread_once
cma_stack_check_limit_np	None
cma_thread_alert	pthread_cancel
cma_thread_bind_to_cpu	None
cma_thread_create	pthread_create
cma_thread_detach	pthread_detach
cma_thread_exit_error	pthread_exit
cma_thread_exit_normal	pthread_exit
cma_thread_get_priority	pthread_getschedparam
cma_thread_get_sched	pthread_getschedparam
cma_thread_get_self	pthread_self
cma_thread_join	pthread_join
cma_thread_set_priority	pthread_setschedparam
cma_thread_set_sched	pthread_setschedparam
cma_time_get_expiration	None
cma_unlock_global	None
cma_yield	None

The HP DCE package (v 1.7) included in HP-UX supports a threads package derived from The Open Group's DCE 1.2.1 version (based on POSIX 1003.4).

The DCE pthreads package does not provide kernel support. It is a purely user-level threads package, hence the name user-space threads.

The mapping of the HP-UX DCE interface threads to the POSIX/UNIX 98 threads is shown in Table 106.

Table 106. Mapping of HP-UX DCE threads to the POSIX/UNIX 98 threads

HP-UX 10.20 and 11.x	**POSIX/UNIX 98**
atfork	pthread_atfork
pthread_cancel_thread	pthread_cancel
pthread_attr_create	pthread_attr_init
pthread_attr_delete	pthread_attr_destroy
pthread_attr_getguardsize_np	pthread_attr_getguardsize
pthread_attr_getinheritsched	pthread_attr_getinheritsched
pthread_attr_getprio	pthread_attr_getschedparam
pthread_attr_getsched	pthread_attr_getschedpolicy
pthread_attr_getstacksize	pthread_attr_getstacksize
pthread_attr_setguardsize_np	pthread_attr_setguardsize
pthread_attr_setinheritsched	pthread_attr_setinheritsched
pthread_attr_setprio	pthread_attr_setschedparam
pthread_attr_setsched	pthread_attr_setschedpolicy
pthread_attr_setstacksize	pthread_attr_setstacksize
pthread_cancel	pthread_cancel
pthread_cond_broadcast	pthread_cond_broadcast
pthread_cond_destroy	pthread_cond_destroy
pthread_cond_init	pthread_cond_init
pthread_cond_signal	pthread_cond_signal
pthread_cond_signal_int_np	None
pthread_cond_timedwait	pthread_cond_timedwait
pthread_cond_wait	pthread_cond_wait
pthread_condattr_create	pthread_condattr_init

HP-UX 10.20 and 11.x	POSIX/UNIX 98
pthread_condattr_delete	pthread_condattr_destroy
pthread_create	pthread_create
pthread_ctxcb_hpux	None
pthread_delay_np	None
pthread_detach	pthread_detach
pthread_equal	pthread_equal
pthread_exit	pthread_exit
pthread_get_expiration_np	get_expiration_time()
pthread_getprio	pthread_getschedparam
pthread_getscheduler	pthread_getschedparam
pthread_getspecific	pthread_getspecific
pthread_is_multithreaded_np	None
pthread_join	pthread_join
pthread_keycreate	pthread_key_create
pthread_lock_global_np	None
pthread_mutex_destroy	pthread_mutex_destroy
pthread_mutex_init	pthread_mutex_init
pthread_mutex_lock	pthread_mutex_lock
pthread_mutex_trylock	pthread_mutex_trylock
pthread_mutex_unlock	pthread_mutex_unlock
pthread_mutexattr_create	pthread_mutexattr_init
pthread_mutexattr_delete	pthread_mutexattr_destroy
pthread_mutexattr_getkind_np	None
pthread_mutexattr_setkind_np	None
pthread_once	pthread_once
pthread_self	pthread_self
pthread_setasynccancel	pthread_setcanceltype

HP-UX 10.20 and 11.x	POSIX/UNIX 98
pthread_setcancel	pthread_setcancelstate
pthread_setprio	pthread_setschedparam
pthread_setscheduler	pthread_setschedparam
pthread_setspecific	pthread_setspecific
pthread_signal_to_cancel_np	None
pthread_testcancel	pthread_testcancel
pthread_unlock_global_np	None
pthread_yield	None
sigprocmask	pthread_sigmask

10.10.7 Limits and default values

This section shows how different values related to the threads library can be obtained. Then, default values are given for attributes.

The following program can be used to extract values from the given system.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <limits.h>
#define w(x) printf("%25.25s %9ld\n", #x, sysconf(x));
main()
{
   w(_SC_CLK_TCK)                   /* # of clock ticks/second */
   w(_SC_VERSION)                   /* POSIX version & revision */
   w(_SC_CHILD_MAX)                 /* Max # of children per process */
   w(_SC_NPROCESSORS_CONF);         /* Number of processors configured. */
   w(_SC_NPROCESSORS_ONLN);         /* Number of processors online. */
   w(_SC_THREADS);                  /* System supports POSIX threads. */
   w(_SC_THREAD_DATAKEYS_MAX);      /* Maximum number of data keys that */
                                       /* can be defined in a process. */
   w(_SC_THREAD_DESTRUCTOR_ITERATIONS);/* Maximum number attempts made */
                                       /* to destroy a thread's */
                                       /* thread-specific data. */
   w(_SC_THREAD_KEYS_MAX);          /* Maximum number of data keys per */
                                    /* process. */
   w(_SC_THREAD_STACK_MIN);         /* Minimum value for the threads */
                                    /* stack size. */
   w(_SC_THREAD_THREADS_MAX);       /* Maximum number of threads within */
```

```
                                    /* a process. */
   w(_SC_REENTRANT_FUNCTIONS);      /* System suuports reentrant */
                                    /* functions (reentrant */
                                    /* functions must be used in */
                                    /* multi-threaded applications). */
   w(_SC_THREAD_SAFE_FUNCTIONS);    /* System supports thread safe */
                                    /* functions. */
   w(_SC_THREAD_ATTR_STACKADDR);    /* System supports the stack */
                                    /* address option for POSIX threads */
                                    /* (stackaddr attribute of threads) */
   w(_SC_THREAD_ATTR_STACKSIZE);    /* System supports the stack size */
                                    /* option for POSIX threads */
                                    /* (stacksize attribute of threads) */
   w(_SC_THREAD_PRIORITY_SCHEDULING); /* System supports the priority */
                                    /* scheduling for POSIX threads. */
   w(_SC_THREAD_PRIO_INHERIT);      /* System supports the priority */
                                    /* inheritance protocol for POSIX */
                                    /* threads (priority inversion */
                                    /* protocol for mutexes). */
   w(_SC_THREAD_PRIO_PROTECT);      /* System supports the priority */
                                    /* ceiling protocol for POSIX */
                                    /* threads (priority inversion */
                                    /* protocol for mutexes). */
   w(_SC_THREAD_PROCESS_SHARED);    /* System supports the process */
                                    /* sharing option for POSIX */
                                    /* threads (pshared attribute of */
                                    /* mutexes and conditions). */
   w(_SC_THREAD_FORKALL);           /* Replicate threads in child proc */

}
```

An example output of this program is:

```
              _SC_CLK_TCK       100
              _SC_VERSION    199506
            _SC_CHILD_MAX       128
     _SC_NPROCESSORS_CONF         2
     _SC_NPROCESSORS_ONLN         2
              _SC_THREADS         1
   _SC_THREAD_DATAKEYS_MAX      450
_SC_THREAD_DESTRUCTOR_ITE         4
      _SC_THREAD_KEYS_MAX       450
     _SC_THREAD_STACK_MIN      8192
    _SC_THREAD_THREADS_MAX    32767
   _SC_REENTRANT_FUNCTIONS        1
_SC_THREAD_SAFE_FUNCTIONS         1
_SC_THREAD_ATTR_STACKADDR         1
```

```
_SC_THREAD_ATTR_STACKSIZE        1
_SC_THREAD_PRIORITY_SCHED       -1
  _SC_THREAD_PRIO_INHERIT       -1
  _SC_THREAD_PRIO_PROTECT       -1
_SC_THREAD_PROCESS_SHARED        1
       _SC_THREAD_FORKALL       -1
```

AIX supports up to 32768 threads in a single process. Each individual pthread requires some amount of process address space, so the actual maximum number of pthreads a process can have depends on the memory model and the use of process address space for other purposes. The amount of memory a pthread needs includes the stack size and the guard region size, plus some amount for internal use.

The user can control the size of the stack with pthread_attr_setstacksize() and the size of the guard region with pthread_attr_setguardsize().

The process address space for 32-bit programs can be increased using the argument -bmaxdata:number at compile time. The value of number may be one of 0x10000000, 0x20000000, ..., or 0x80000000. This will allocate between one and eight additional segments of 256 MB memory for the process. Table 107 lists the default values.

Table 107. Default values for pthreads attributes in AIX 5L

Attribute	**AIX 5L**	**HP-UX 11i**	**Solaris 8**	**Tru64**
scope	PTHREAD_SCOPE _PROCESS	PTHREAD_SCOPE _SYSTEM	PTHREAD_SCOPE _PROCESS	PTHREAD_SCOPE _PROCESS
detachstate	PTHREAD_ CREATE_ JOINABLE	PTHREAD_ CREATE_ JOINABLE	PTHREAD_ CREATE_ JOINABLE	PTHREAD_ CREATE_ JOINABLE
stackaddr	N/A	NULL	NULL	NULL
stacksize	96 KB PTHREAD_ STACK_MIN	64 KB	NULL (1 MB for 32 bit processes, 2 MB for 64 bit processes)	
priority	1	N/A	0	19
inheritsched	PTHREAD_ INHERIT_SCHED	PTHREAD_ INHERIT_SCHED	PTHREAD_ EXPLICIT_SCHED	PTHREAD_ INHERIT_SCHED
schedpolicy	SCHED_OTHER	SCHED_ TIMESHARE	SCHED_OTHER	SCHED_OTHER
guardsize	PAGESIZE	PAGESIZE	PAGESIZE	PAGESIZE

Attribute	AIX 5L	HP-UX 11i	Solaris 8	Tru64
cancelability state	PTHREAD_ CANCEL_ENABLE	PTHREAD_ CANCEL_ENABLE	PTHREAD_ CANCEL_ENABLE	PTHREAD_ CANCEL_ENABLE
cancelability type	PTHREAD_ CANCEL_ DEFERRED	PTHREAD_ CANCEL_ DEFERRED	PTHREAD_ CANCEL_ DEFERRED	PTHREAD_ CANCEL_ DEFERRED

10.10.8 Inspecting a process and its kernel threads

AIX provides the `ps` command for showing the current process status. Setting the appropriate flags, we can also show the kernel threads information. To display information about all processes and kernel threads on AIX 5L, enter:

```
ps -emo THREAD
```

To display information about a certain user and associated kernel threads, enter:

```
ps -mo THREAD -u root
```

The output is similar to:

```
USER    PID  PPID     TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND

root      1     0       - A    0  60  1        -   200003      -   - /etc/init
   -      -     -     259 S    0  60  1        -   410410      -   - -
root   2714  4434       - A    0  60  1        -   240001      -   - /usr/sbin/inetd
   -      -     -    4977 S    0  60  1        -    18400      -   - -
root   3482     1       - A    0  60  1   1b3234    40401      -   - /usr/lib/errdemon
   -      -     -    4015 S    0  60  1   1b3234    10400      -   - -
root   3642     1       - A    1  60  1 32dcb298   240001      -   - /usr/sbin/syncd 60
   -      -     -    3713 S    1  60  1 32dcb298    10400      -   - -
root   3974  4434       - A    0  60  1 70214114   240001      -   - /usr/sbin/writesrv
   -      -     -    4525 S    0  60  1 70214114      400      -   - -
root   4214  4434       - A    0  60  1        -   240001      -   - /usr/sbin/syslogd
   -      -     -    5235 S    0  60  1        -    18400      -   - -
daemon 8520 4434       - A    0  60  3        -   240001      -   - /usr/sbin/rpc.statd
   -      -     -    8779 Z    0  60  1        -   c00001      -   - -
   -      -     -    9041 Z    0  60  1        -   c00001      -   - -
   -      -     -    9295 S    0  60  1        -   418400      -   - -
root   8776  4434       - A    0  60  1 c0043100   240001      -   - /usr/sbin/qdaemon
   -      -     -   10583 S    0  60  1 c0043100    10400      -   - -
```

The columns are defined as:

USER	The login name of the process owner
PID	The process ID of the process
PPID	The process ID of the parent process
TID	The thread ID of the kernel thread
ST	The state of the process or kernel thread
CP	The CPU utilization of the process or kernel thread
PRI	The priority of the process or kernel thread
SC	The suspend count of the process or kernel thread
WCHAN	The wait channel of the process or kernel thread
F	The flags of the process or kernel thread
TT	The controlling terminal of the process
BND	The CPU to which the process or kernel thread is bound
CMD	The command being executed by the process

In the example output shown above, we can see that process 8520 has three kernel threads, two in CANCLED state and one in SLEEPING state.

10.11 Example: The Mandelbrot set

The Mandelbrot set is named after the mathematician Benoit B. Mandelbrot. This section uses the computation of a Mandelbrot set as the basis for an example program using POSIX threads.

The example consists of five small C-programs. Error checking has been kept to a minimum (see Section 10.1.1.3, “Example of a POSIX threaded program” on page 310). The source code of the programs is listed in Appendix A, “Sample programs” on page 451. The descriptions of these example programs is given in Table 108.

Table 108. Description of example programs

File name	Description
mandelbrot1.c	Prints Figure 67 on page 392.
mandelbrot2.c	Computes set sequentially, one horizontal line at the time, no use of pthreads.
mandelbrot3.c	Computes set using one thread pr. horizontal line.
mandelbrot4.c	Computes set using one thread per horizontal line, which may create more threads for a specific horizontal line, if it detects that the computation has taken more than three seconds.
mandelbrot5.c	Computes set using one thread per horizontal line, which may create more threads for a specific horizontal line, if it detects that the computation has taken more than three seconds and a CPU appears to be more than 20 percent idle.

```
:::::::::::::::::::::::::::::::::::::::::::::::::::::=======================o++====================:::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::============================++o++==+==================::::::::::
:::::::::::::::::::::::::::::::::::::::::::=================================+++o+o+++===================::::::::
::::::::::::::::::::::::::::::::::::::====================================++++ooo+++======================::::::
::::::::::::::::::::::::::::::::::=====================================+++xoooo  oo+oo+=====================::::
::::::::::::::::::::::::::::::========================================+++++o        o+++======================::
::::::::::::::::::::::::::=====================================+++++++++++oO       o++++++=====================:
::::::::::::::::::::::====================================++xooo+++++Ox+OooooO   OooooO++oo++++++o+=============
:::::::::::::::::::=======================================++++x  OoXox                    oo+++xoooo++==========
::::::::::::::::========================================++++++O                              o   @o++===========
:::::::::::::=========================================++xooooX                                  o++++===========
:::::::::::=====================+++========++======+++++++x                                       o+++o=========
::::::::========================++o++++++++++++++++++++xX                                            Oo+========
::::::==========================+++++oxx+oox oooo++++++X                                           x+++=========
:::===========================+++++++o            oo++oo                                           : o+=========
::========================++++++++oxX               Ooo                                            O++==========
========================+++++o+oooo                   x                                            O+===========
============++++++++++++++++oo    @                   O                                          +++============
============++++++++++++++++oo    @                   O                                          +++============
========================+++++o+oooo                   x                                            O+===========
::========================++++++++oxX               Ooo                                            O++==========
:::===========================+++++++o            oo++oo                                           : o+=========
::::::==========================+++++oxx+oox oooo++++++X                                           x+++=========
::::::::========================++o++++++++++++++++++++xX                                            Oo+========
:::::::::::=====================+++========++======+++++++x                                       o+++o=========
:::::::::::::=========================================++xooooX                                  o++++===========
::::::::::::::::========================================++++++O                              o   @o++===========
:::::::::::::::::::=======================================++++x  OoOox                    oo+++xoooo++==========
::::::::::::::::::::::====================================++xooo+++++Ox+OooooO   OooooO++oo++++++o+=============
::::::::::::::::::::::::::=====================================+++++++++++oO       o++++++=====================:
::::::::::::::::::::::::::::::========================================+++++o        o+++======================::
::::::::::::::::::::::::::::::::::=====================================+++xoooo  oo+oo+=====================::::
::::::::::::::::::::::::::::::::::::::====================================++++ooo+++======================::::::
:::::::::::::::::::::::::::::::::::::::::::=================================+++o+o+++===================::::::::
::::::::::::::::::::::::::::::::::::::::::::::::============================++o++==+==================::::::::::
```

Figure 67. Output from mandelbrot1.c, 35 horizontal lines

We see the results of running the programs on an IBM @server pSeries 680 with 24 CPUs as follows.

All the programs, except mandelbrot1.c, each have several parameters, which can be altered. The code contains the following `#defines`:

```
#define MAX_ITERATION 262144
#define MAX_LENGTH 100
#define X_MIN -2.1
#define Y_MIN -1.1
#define X_MAX 0.7
#define Y_MAX 1.1
#define RESOLUTION 35 /* vertical resolution, horizontal is then derived */
```

Table 109 shows the results of various runs with the specified values. They are all run with the defaults settings for the pthread library on AIX 5L, that is, process contention-scope and an AIXTHREAD_MNRATIO of 8:1.

Table 109. Timing data for mandelbrot programs

Program	MAX_ITERATION	RESOLUTION	Time
mandelbrot2	262144	35	real 0m40.69s user 0m40.68s sys 0m0.00s

Program	MAX_ITERATION	RESOLUTION	Time
mandelbrot3	262144	35	real 0m8.45s user 0m41.62s sys 0m0.02s
mandelbrot4	262144	35	real 0m7.27s user 0m43.30s sys 0m0.02s
mandelbrot5	262144	35	real 0m6.87s user 0m43.46s sys 0m0.07s
mandelbrot2	524288	50	real 2m46.38s user 2m46.36s sys 0m0.01s
mandelbrot3	524288	50	real 0m25.00s user 2m49.71s sys 0m0.01s
mandelbrot4	524288	50	real 0m13.77s user 3m5.13s sys 0m0.04s
mandelbrot5	524288	50	real 0m13.24s user 3m8.59s sys 0m0.23s
mandelbrot2	1048576	100	real 22m13.66s user 22m13.51s sys 0m0.01s
mandelbrot3	1048576	100	real 2m43.85s user 22m42.24s sys 0m0.00s
mandelbrot4	1048576	100	real 1m2.72s user 38m45.70s sys 0m0.17s
mandelbrot5	1048576	100	real 1m6.22s user 39m13.00s sys 0m2.58s

The command `ps -emo THREAD` can be run while the programs run in order to see the kernel threads.

The main difference between mandelbrot4 and mandelbrot5 is that the latter tries to avoid starting threads uncritically. The running times are similar, but fewer threads are started, relieving pressure from the library scheduler. In this example, it imposes a minor additional amount of computation time.

Figure 68 shows how the real time is lower for mandelbrot4 and mandelbrot5, due to the fact that idle processor time is exploited. However, we also see that the extra layer of scheduling has an impact on the user time.

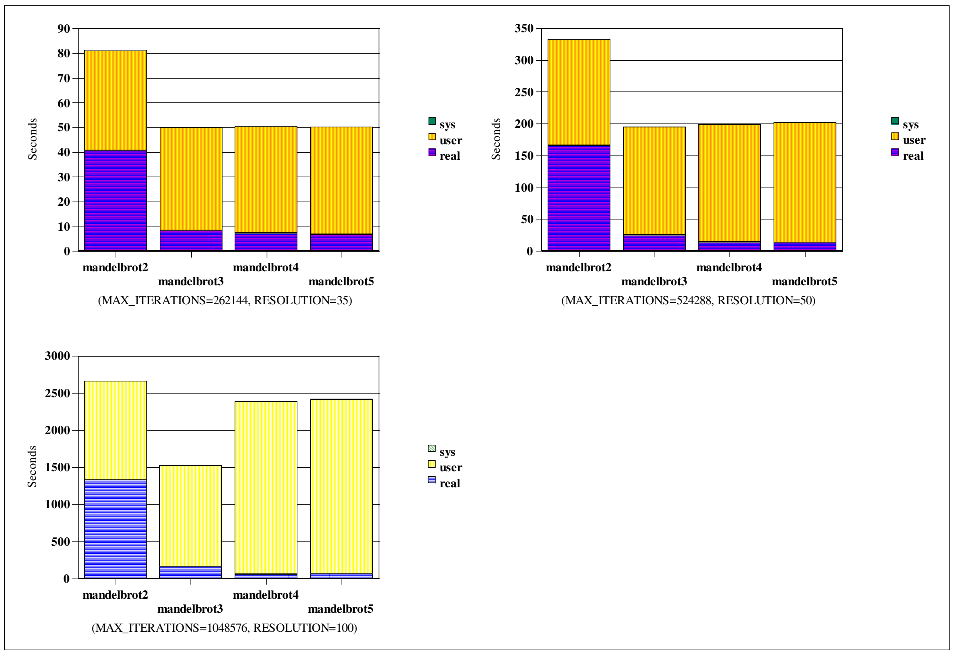


Figure 68. Charts of execution time

The previous example shows that the use of threads should be carefully considered. For example, the computation of the Mandelbrot set, as presented here, has the property that the computation can be subdivide into independent computations, requiring no synchronization, expect for printing out the result. Other problems might be characterized as interdependencies between intermediate results and the impossibility to subdivide the computation beyond a certain level of granularity.

The example showed, that threading does impose an overhead in user time but allows the exploitation of idle processors.

Other tests could be performed, for example, altering some of the environment variables mentioned in Section 10.8, “Tuning” on page 367.

Also, the optimization options of the compiler have not been used. For this example, they will yield impressive results, due to the nature of the problem.

10.11.1 References

The following references contain additional information on threads:

- IEEE Standards Online Subscriptions, found at:

  `http://standards.ieee.org/catalog/olis/index.html`

  This Web site requires registration, which can be done online. Use search criteria like 9945-1 or 1003.1.

- *The Single UNIX Specification, Version 2*, found at:

  `http://www.opengroup.org/onlinepubs/7908799/toc.htm`

- *POSIX Programmer's Guide: Writing Portable Unix Programs With the POSIX*, by Lewine, et al.

Chapter 11. C++ templates

Templates are an area of the C++ language that provide a great deal of flexibility for developers. The recent ANSI C++ standard defines the language facilities and features for templates. Unfortunately, the standard does not specify how a compiler should implement templates. This means that there are sometimes significant differences between the methods used to implement templates in compiler products from different vendors.

11.1 Using C++ templates

Developers porting C++ code that uses templates to the AIX 5L platform sometimes have problems with the implementation model. The main problems experienced are:

- Long compile and link times
- Linker warnings of duplicate symbols
- Increase in code and executable size

All of these above problems are generally caused by the inefficient use of the AIX 5L implementation of templates. The number of problems experienced will depend on the platform the code is being ported from and the template implementation method used on that platform. Sometimes the problems can be fixed on AIX 5L by simply adding a few compiler options. In other instances, the code layout needs to be changed in order to utilize the most efficient implementation method on AIX 5L. In most of these rare cases, the code changes are backwards compatible with the original platform the code is being ported from. This is very important for developers who maintain a single source tree that must compile correctly on multiple platforms.

11.2 AIX 5L template implementations

The template mechanism provides a way of defining general container types, such as list, vector, and stack, where the specific type of the elements is left as a parameter. Two types of templates can be defined:

Class templates	Specify how individual classes can be constructed.
Function templates	Specify how individual functions can be constructed.

Regardless of the type of template being used, the code is essentially split into three parts:

Template declaration	This is the part of the source code that declares the template class or function. It does not necessarily contain the definition of the class or function, although it may optionally do so. For example, a template declaration may describe a Stack template class, as shown in Figure 69.
Template definition	This portion of code is the definition of the template function itself or the template class member functions. For example, using the Stack class template, this portion of code would define the member functions used to manipulate the stack, as shown in Figure 70 on page 399.
Template instance	The template instance code is generated by the compiler for each instance of the template. For example, this would be the code to handle a specific instance of the stack template class, such as a stack of integer values.

The difference between the components is that the template declaration must be visible in every compilation unit that uses the template. The template definition and code for each instance of the template need only be visible once in each group of compilation units that are linked together to make a single executable.

```
template <class T> class stack
{
private:
   T* v;
   T* p;
   int sz;
public:
   stack( int );
   ~stack();
   void push( T );
   T pop();
};
```

Figure 69. Stack template declaration

```
template <class T> stack<T>::stack( int s )
{
        v = p = new T[sz=s];
}

template <class T> stack<T>::~stack()
{
        delete [] v;
}

template <class T> void stack<T>::push( T a )
{
        *p++ = a;
}

template <class T> T stack<T>::pop()
{
        T ret = *p;
        p--;
        return ret;
}
```

Figure 70. Stack template member function definition

11.2.1 Generated function bodies

When you use class templates and function templates in your program, the compiler automatically generates function bodies for all template functions that are instantiated. The compiler follows four basic rules to determine when to generate template functions. The compiler applies the rules in the following order:

1. If a template declares a function to have internal linkage, the function must be defined within the same compilation unit. The compiler generates the function with internal linkage, and it is not shared with other compilation units. This is the case if the template class has in-line member functions.
2. If a template function is instantiated in a compilation unit, but it is not declared to have internal linkage, the compiler looks for a definition of the function in the same compilation unit. If a definition is found, the compiler generates a function body in the same compilation unit.
3. If a template function is instantiated in a compilation unit, and the function is not defined in the same compilation unit, but certain other conditions are met, the compiler generates the necessary function definitions during a

special pre-link phase of the compilation. This is the case when the -qtempinc option is in use.

4. If none of the preceding rules applies, the compiler does not generate the definition of the template function. It must be defined in another compilation unit.

11.3 Simple code layout method

The simplest method of using template code is to include both the declaration and definition of the template in every compilation unit that uses instances of the template. From a code layout point of view, this is very easy, because the template declaration and definition can be kept in a single header file. Using the stack example, the code in Figure 69 on page 398 and Figure 70 on page 399 would be combined into a single header file, for example, stack.h, which is then included by every compilation unit that wishes to use the template. Alternatively, the header file for a template declaration can include the source file that contains the template definition. Using the stack template example, the header file, stack.h, would `#include` the source file, stack.C.

There are a number of disadvantages to using this method. Some of them can be overcome; others cannot.

11.3.1 Disadvantages of the simple method

The first disadvantage of the simple method is that using the header files can become complicated, particularly when other header files need to declare an instance of the template. In order to do this, they must `#include` the stack.h file, which potentially leads to multiple `#include`s of the file, resulting in multiple definitions of the member functions. This problem can be fixed with the addition of preprocessor macros in the header file to protect against multiple `#include` operations. For example:

```
#ifndef stack_h
#define stack_h
....
....declaration and definition of stack template
....
#endif
```

Using the macros shown above, the contents of the header file will only appear once in the compilation unit, regardless of the number of times the file is included. This resolves the problems of multiple definitions within a compilation unit.

11.3.1.1 Template code bloat

The second disadvantage is that the code for each template instance will potentially appear multiple times in the final executable, resulting in the twin problems of large executable size and multiple symbol definition warnings from the linker.

As an example, consider an executable made up of two compilation units, main.C and functions.C. If both compilation units include the stack.h header file declare variables of the type stack<int>, then after the first stage of compilation, both object files, main.o and functions.o, will contain the code for the member functions of the stack<int> class. When the system linker parses the object files to create the final executable, it cannot remove the duplicate symbols because, by default, each compilation unit is treated as an atomic object by the linker. This results in duplicate symbol linker warnings and a final executable that contains redundant code.

The size of the final executable can be reduced by using the compiler option -qfuncsect, when compiling all of the source code modules. This option causes the compiler to slightly change the format of the output object files. Instead of creating an object file, which contains a single code section (CSECT), which must be treated by the system linker as an atomic unit, the compiler creates an object file, where each function is contained in its own CSECT. This means that the object files created are slightly larger than their default counterparts, because they contain extra format information, in addition to the executable code. This option does not remove the linker warnings because at link time, there are still multiple symbol definitions. The benefit of this option is that the linker can discard multiple, identical function definitions by discarding the redundant CSECTs, resulting in a smaller final executable. When the -qfuncsect option is not used, the compiler cannot discard the redundant function definitions if there are other symbols in the same CSECT that are required in the final executable.

Refer to Section 11.6, “Virtual functions” on page 409 for information on another potential cause of C++ code bloat.

11.3.1.2 Template compile time

The use of the -qfuncsect option reduces the code size of the final executable. It does not resolve the other disadvantage of using this method (of longer than required compile times). The reason for this is that each compilation unit contains the member functions for the templates that it instantiates. Using an extreme example with the stack class, consider the situation where an application is built from 50 source files, and each source file instantiates a stack<int> template. This means the member functions for the class are generated and compiled 50 times, yet the result of 49 of those compiles are discarded by the linker because they are not needed. In the example used here, the code for the stack class is trivial, so in absolute terms, the time saved would be minimal. In real life situations, where the template code is complex, the time that can be saved when compiling a large application is considerable.

Because not all of the disadvantages of the simple template method can be overcome, it is only recommended for use when experimenting with templates. An alternative method can be used, which solves all of the problems of the simple method and scales very well for large applications.

11.4 Preferred template method

The preferred method of template instantiation on AIX 5L basically means letting the compiler decide which template code to instantiate as a final step in the compile and link process. This solves the long compile time disadvantage of the simple template method, because the compiler only needs to compile each template instance once.

This method requires that the declaration and definition of the template are kept in separate files. This is because only the template declaration must be included in every compilation unit that uses the template. If the definition of the template were also in the header file, it would also be included in the source file and thus compiled, resulting in a situation similar to that in the simple method.

The preferred template model can also benefit from the use of the -qfuncsect compiler option, because it means the linker can discard code sections that are not referenced in the final executable.

The template declaration should be left in the header file, as in the simple template method. The definition of the template member functions needs to be in a file with the same base name as the header file, but with a .c (lower case C) file name extension.

Note

By default, the file containing the template definition code must have the same name as the template declaration header file, but with a file name extension of .c (lowercase c), even though this extension normally indicates a C language source file. It must also exist in the same directory as the template declaration header file. If the template definition file is not in the same directory, has a different base name, or has a different file name extension (such as .C, .cxx, or .cpp, which are normally used for C++ source files), then the compiler will not detect the presence of the template code to be used with the template declaration header file.

Using the stack template example introduced earlier, the template declaration shown in Figure 69 on page 398 would be in the file stack.h, while the template definition code shown in Figure 70 on page 399 would be in the file stack.c in the same directory. If the template definition code file was named stack.cxx or stack_code.c, then the compiler will not associate the file with the template declaration in the stack.h header file.

The name of the template definition file can be changed, if desired, using the implementation `pragma` directive as follows:

```
#pragma implementation(string-literal)
```

where string-literal is the path name for the template definition file enclosed in double quotes. For example, if the stack template definition code were to be stored in the file stack_code.cxx, then the stack.h header file would have the following directive:

```
#pragma implementation("stack_code.cxx")
```

Once the structure of the source code has been altered to conform to the required layout, the templates can be used in the preferred way.

11.4.1 The -qtempinc option

The -qtempinc option is used when compiling source code that instantiates templates. When no directory is specified with the option, the compiler will create a directory called tempinc in the current directory. For example:

```
xlC main.C -qtempinc
```

The user may optionally specify the name of a directory to use for storing the information on the templates to be generated. This allows the same tempinc directory to be used when creating an executable that consists of object files that are compiled in different directories. For example:

```
xlC -c file1.C file2.C -qtempinc=../app1/templates
cd ../app1
xlC -o app1 main.C ../src/file1.o ../src/file2.o -qtempinc=./templates
```

The tempinc directory is used to store information about the templates that are required to be generated. When invoked with the `-qtempinc` option, the compiler collects information about template instantiations and stores the information in the tempinc directory. As the last step of the compilation before linking, the compiler generates the code for the required template instantiations. It then compiles the code and includes it with the other object files and libraries that are passed to the linker to create the final executable.

If the compiler detects a code layout structure that enables the preferred template method to be used, it will automatically enable the -qtempinc option, even if it was not specified on the command line. This causes the template instantiation information to be stored in the tempinc directory. If you want to specify a different directory, you should explicitly use the -qtempinc=dirname option on the command line. If you want to prevent the compiler from automatically generating the template information, which may be the case when creating a shared object, then use the -qnotempinc option. Refer to Section 11.5, “Shared objects with templates” on page 406 for more information on the use of the -qnotempinc option when creating shared objects.

One important point to note about the -qtempinc option is that you should use the same value when compiling all compilation units that will be linked together. In other words, do not compile half of the application with -qtempinc, and the other half with -qtempinc=dirname. Only one tempinc directory can be specified on the final C++ compile line that is used to link the application, which means that half of the template instance information will be missing. If more than one tempinc option is specified on the command line, the last one encountered will prevail.

11.4.2 Contents of the tempinc directory

The compiler generates a file in the tempinc directory for each template header file that has templates instantiated. The file has the same name as the header file, but with a .C (uppercase C) file name extension. The compiler generates the file when it detects the first instantiation of a template that is declared in the header file with the same name. Information on the subsequent instances of the template is added to the file.

As the final step of the compilation before linking, the compiler compiles all of the files in the tempinc directory and passes the object files to the linker along with the user specified files.

The contents of a template information file are as follows:

```
/*0965095125*/#include "/redbooks/examples/C++/stack.h"          1
/*0000000000*/#include "/redbooks/examples/C++/stack_code.cxx"   2
template stack<int>::stack(int);                                 3
template stack<int>::~stack();                                   4
template void stack<int>::push(int);                             5
template int stack<int>::pop();                                  6
```

The line numbers at the end of each line have been added for reference purposes. The code on line 1 includes the header file that declares the template. The comment at the start of the line is a time stamp and is used by the compiler to determine if the header file has changed, which would require the template instance information file to be recompiled.

The code on line 2 includes the template implementation file that corresponds to the header file in line 1. A time stamp consisting of all zeros indicates that the compiler should ignore the time stamp. The file may include other header files that define the classes that are used in template instantiations. For example, if there was a user defined class Box, and the compiler detected an instantiation of stack<Box>, then the header file that defines the class Box would be included in the instance information file.

The subsequent lines in the example shown above cause the individual member functions to be instantiated.

11.4.3 Forcing template instantiation

You can, if you wish, structure your program so that it does not use automatic template instantiation. In order to do this, you must know which template classes and functions need to be instantiated.

The `#pragma` define directive is used to force the instantiation of a template, even if no reference is made to an instance of the generated template. For example:

```
#pragma define(stack<double>);
```

This, however, means that the template implementation file needs to be included in the compilation units that have the `#pragma` define directives, which results in the same disadvantages of the simple template method described in Section 11.3, “Simple code layout method” on page 400.

An alternative to this is to manually emulate the process used by the compiler to automatically create the appropriate template instances. Using the stack class as an example, the following compilation unit could be used to force the creation of the desired stack template classes, even though no objects of those types are referenced in the source code:

```
#include "/redbooks/examples/C++/stack.h"
#include "/redbooks/examples/C++/stack_code.cxx"
#include “/redbooks/examples/C++/Box.h” // definition of class Box
#pragma define(stack<int>);
#pragma define(stack<Box>);
#pragma define(stack<char>);
#pragma define(stack<short>);
```

This type of method will be useful when creating shared objects with the `makeC++SharedLib` command. Users of the VisualAge C++ Professional for AIX 5L Version 5 compiler should use the -qmkshrobj option instead. Refer to Section 11.5, “Shared objects with templates” on page 406 for more information.

11.5 Shared objects with templates

Templates are usually declared in a header file. Each time a template is used, code is generated to instantiate the template with the desired parameters. Most C++ compilers work with a template repository. No template code is generated at compile time; the compiler just remembers where the template code came from. Then, at link time, as the compiler/linker puts all parts together, it notices which templates actually need to be generated. The code is then produced, compiled, and linked into the application.

This becomes a problem when using templates with shared libraries, where no actual linking takes place. So, one must make sure that the template code is generated when producing the shared library.

Therefore, one should keep track of compilation and inclusion of template instantiations. This would mean that one has to manually keep track of all the template instantiation and address them during the linking phase.

It is here that the VisualAge C++ Professional for AIX Version 5 compiler has a noticeable improvement over previous versions of C++ compilers for AIX. The compiler, like the `makeC++SharedLib` command, can be used to create a shared object from object files using the -qmkshrobj option.

This option, together with the -qtempinc option, should be used in preference to the `makeC++SharedLib` command when creating a shared object that uses templates. The advantage of using these options instead of `makeC++SharedLib` is that the compiler will automatically include and compile the template instantiations in the tempinc directory.

11.5.1 Templates and makeC++SharedLib

The `makeC++SharedLib` command is supplied with the IBM C++ command line compilers for the AIX 5L platform. The command is a shell script that gathers the supplied input and then calls the linker to create the shared object. While this script is available on AIX 5L, the suggested method of building a shared library that uses template code is described in Section 11.5.2, “Templates and -qmkshrobj” on page 408.

When creating a shared object that uses templates, the `makeC++SharedLib` command needs to somehow find information on the templates that are to be instantiated. Because the script calls the linker, and not the compiler, it does not look at the contents of the tempinc directory. This means the method of creating a shared object that uses templates relies on either using the simple template method code layout, as described in Section 11.3, “Simple code layout method” on page 400, or forcing templates to be instantiated, as described in Section 11.4.3, “Forcing template instantiation” on page 405.

The best method to use will depend on the circumstances. Using the simple code layout method means that all the required templates are automatically generated. However, it also comes with the disadvantages of slower compile times and larger code size. Forcing the templates to be instantiated is better from both the code size and compile time aspect, but it does mean that the user needs to maintain files that instantiate the required templates.

Suppose you want to create a shared object from the following two source files, which use the preferred code layout method.

The file source1.C contains the following code:

```
#include “stack.h”
stack<int> counter1;
void function1(int a)
{
   counter1.push(a);
}
```

The file source2.C contains the following code:

```
include “stack.h”
stack<int> counter2;
void function2(int a)
```

```
{
   counter2.push(a);

}
```

Using the `makeC++SharedLib` command, an attempt is made to create a shared object, as follows:

```
xlC -c source1.C source2.C
/usr/vacpp/bin/makeC++SharedLib -o shr1.o -p0 source1.o source2.o
ld: 0711-317 ERROR: Undefined symbol: .stack<int>::stack(int)
ld: 0711-317 ERROR: Undefined symbol: .stack<int>::~stack()
ld: 0711-317 ERROR: Undefined symbol: .stack<int>::push(int)
ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more
information.
```

The command failed, and based on the output, it is easy to see that the required template functions have not been instantiated. At this point, note that because the code uses the preferred code layout method, the compiler has, in fact, automatically created the file tempinc/stack.C, which, if compiled, would supply the required template definitions. You can, if you wish, copy this file and make it an explicit compilation unit as part of your source code. In this case, that would mean adding the following command to the sequence:

```
xlC -c tempinc/stack.C -o stack.o
```

The object, file stack.o, would then be passed to the makeC++SharedLib command along with source1.o and source2.o.

11.5.2 Templates and -qmkshrobj

Users of the VisualAge C++ Professional for AIX Version 5 compiler should use the -qmkshrobj option in preference to the `makeC++SharedLib` command when creating a shared object. Because the option is a compiler option, it will automatically look in the tempinc directory (or the directory specified with the -qtempinc=<dirname>. option) for the automatically generated template instance information. Using the same source files described in Section 11.5.1, “Templates and makeC++SharedLib” on page 407, the following commands can be used to create the shared object:

```
xlC -c source1.C source2.C
xlC -qmkshrobj -o shr1.o source1.o source2.o
```

This time the command works correctly, because the compiler looks in the tempinc directory. Remember to use the same -qtempinc option (if any) that was used when compiling the modules being used to create the shared object.

This option solves the problems associated with creating shared objects that use template classes and functions. If you want to create a shared object that contains pre-instantiated template classes and functions for use by other developers, then you can create an additional compilation unit that explicitly defines the required templates using the `#pragma` define directive.

11.6 Virtual functions

In general, when writing C++ code, you should try and avoid the use of virtual functions. They are normally encoded as indirect function calls, which are slower than direct function calls.

Usually, you should not declare virtual functions in-line. If all virtual functions in a class are in-line, the virtual function table and all the virtual function bodies will be replicated in each compilation unit that uses the class. The disadvantage to this is that the virtual function table and function bodies are created with internal linkage in each compilation unit. This means that even if the -qfuncsect option is used, the linker cannot remove the duplicated table and function bodies from the final executable. This can result in very bloated executable size.

Chapter 12. Test and debug

This chapter presents two debugging tools. For details on all the functions and features of these tools, please consult the online documentation. In the following sections, we will walk the reader through some examples, illustrating the use of these tools.

12.1 dbx

In this section we will use `dbx` to debug two pieces of code. The dbx command is the tty-based symbolic debugger supplied with AIX 5L on Power platforms. For more information on `dbx`, please see *General Programming Concepts: Writing and Debugging Programs*, which can be found in the AIX 5L online documentation.

12.1.1 Small example

Consider the following code:

```
$ cat debugstrings.c
int main(int argc, char **argv)
{

        char mystring1[10] = "foo";
        char *mystring2;

        mystring2 = (char *)malloc(sizeof(char)*(long)10);

        strcpy(mystring2, "bar\n\0");

        printf("%s%s", mystring1, mystring2);

}
$
```

The code is compiled using and run with both -q32 and -q64 options:

```
$ cc -q32 -o debugstrings debugstrings.c
$ debugstrings
foobar
$ cc -q64 -o debugstrings debugstrings.c
$ debugstrings
Segmentation fault(coredump)
$
```

We then recompile (using the debug option), re-run the program, and find the core file:

```
T$ cc -g -q64 -o debugstrings debugstrings.c
$ debugstrings
Segmentation fault(coredump)
$ ls -rt core* | tail -1
core.21184.24174837
$
```

Note that on AIX 5L, a core file is not just called core. The format is:

`core.pid.ddhhmmss`

where:

- pid is the process ID of the process that caused the core.
- dd indicates the day of the month.
- hhmmss is a time stamp indicating the hours, minutes and seconds.

We then start the debugger, contemplate the error message, and quit. The debugger's prompt is `(dbx)`:

```
$ dbx debugstrings core.21184.24174837
Type 'help' for help.
reading symbolic information ...
[using memory image in core.21184.24174837]

Segmentation fault in strcpy.strcpy [debugstrings] at 0x1000005d8
0x1000005d8 (strcpy+0xb8) 9ce50001       stbu   r7,0x1(r5)
(dbx) q
```

It seems to be a problem in strcpy. We then step through the program and use the `whatis` and `print` commands, we see:

```
 $ dbx debugstrings
Type 'help' for help.
reading symbolic information ...
(dbx) step
stopped in main at line 4
    4           char mystring1[10] = "foo";
(dbx) step
stopped in main at line 7
    7           mystring2 = (char *)malloc(sizeof(char)*(long)10);
(dbx) step
stopped in main at line 9
    9           strcpy(mystring2, "bar\n\0");
(dbx) step

Segmentation fault in strcpy.strcpy [debugstrings] at 0x1000005d8
0x1000005d8 (strcpy+0xb8) 9ce50001       stbu   r7,0x1(r5)
(dbx) whatis mystring2
unsigned char *mystring2;
(dbx) print mystring2
(invalid char ptr (0x00000000100007d0))
(dbx)
```

It appears that mystring2 points to an invalid address. Let us rerun the string and see, using the `print` command, how it gets this value in the first place:

```
$ dbx debugstrings
Type 'help' for help.
reading symbolic information ...
(dbx) step
stopped in main at line 4
    4           char mystring1[10] = "foo";
(dbx) print mystring1
""
(dbx) print mystring2
(invalid char ptr (0xbadc0ffee0ddf00d))
(dbx) step
stopped in main at line 7
    7           mystring2 = (char *)malloc(sizeof(char)*(long)10);
(dbx) print mystring1
"foo"
(dbx) print mystring2
(invalid char ptr (0xbadc0ffee0ddf00d))
(dbx) step
stopped in main at line 9
    9           strcpy(mystring2, "bar\n\0");
(dbx) print mystring2
(invalid char ptr (0x00000000100007d0))
(dbx) print &mystring1[0]
0x0ffffffffffff50
```

It seems that the malloc routines assigns it an address in segment 1. This is the main program text segment and is not writable by the program.

Let us run the 32-bit version and print the address values:

```
$ cc -g -q32 -o debugstrings debugstrings.c
$ dbx debugstrings
Type 'help' for help.
reading symbolic information ...
(dbx) step
stopped in main at line 4
    4           char mystring1[10] = "foo";
(dbx) step
stopped in main at line 7
    7           mystring2 = (char *)malloc(sizeof(char)*(long)10);
(dbx) step
stopped in main at line 9
    9           strcpy(mystring2, "bar\n\0");
(dbx) print &mystring1[0]
0x2ff22bb0
(dbx) print &mystring2[0]
0x20000928
(dbx)
```

We see that the strings are allocated in segment 2.

Section 3.8, “Pointer assignment and arithmetic” on page 48 leads us to suspect that the address malloc should return is being truncated. Our suspicion is confirmed by the following:

```
$ cc -qwarn64 -o debugstrings debugstrings.c
"debugstrings.c", line 7.21: 1506-745 (I) 64-bit portability: possible incorrect pointer
through conversion of int type into pointer.
```

It looks like a missing function prototype for the malloc routine. After including the header file stdlib.h, the code runs without problems:

```
$ cat debugstrings.c
#include <stdlib.h>

int main(int argc, char **argv)
{

        char mystring1[10] = "foo";
        char *mystring2;

        mystring2 = (char *)malloc(sizeof(char)*(long)10);

        strcpy(mystring2, "bar\n\0");

        printf("%s%s", mystring1, mystring2);

}
$ cc -g -q64 -qwarn64 -o debugstrings debugstrings.c
$ debugstrings
foobar
$
```

12.2 debug_message.c and dbx

In this section, we will use `dbx` to debug the code `debug_message.c`:

```
/*
* Program debug_message
*
* This program has been modified to exhibit a few bugs that can be
* investigated with the debugger.
*
* The program generates a number of child processes using fork and
* communicates between them using messages.
*
*/
#include <stdio.h>      /* Needed for printf */
#include <sys/shm.h>    /* Needed for shmget */
#include <sys/types.h>  /* Needed for fork */
#include <unistd.h>     /* Needed for fork */
#include <sys/msg.h>    /* Needed for msgget... */
#include <string.h>     /* Needed for strncmp */

/* maximum message size */
#define MAX_MSG_SIZE    64

#define NKIDS   3       /* How many children? */
#define NUM_OF_MSGS 2   /* How many predefined messages are there? */

struct mess {
    mtyp_t  mtype;
    char    mtext[MAX_MSG_SIZE];
} the_word;

char    *msgs[NUM_OF_MSGS]  = {"cd /pub","more beer"};

int identity    = -1;           /* Define and initialize */
int ix  = -1;                   /* Define and initialize */

main()
{

    pid_t   pid = -1;           /* Define and initialize */
    int     nkids = -1;         /* Define and initialize */
    int     child = 0;          /* Set child to false */
    int     parent = 0;         /* Set parent to false */
    key_t   mkey    = 0xf00;    /* Define and initialize */
    const   int shmsize = 8192; /* Define and initialize */
    int     shmid   = -1;       /* Define and initialize */
    int itemp   = -1;
    int msgid   = -1;

    /* Initialise a message queue */
    itemp   = IPC_CREAT|S_IRUSR|S_IWUSR;
    if((msgid=msgget(IPC_PRIVATE,itemp))==-1) {
        perror("msgget #1"); exit(1); }

    printf("Parent >  Making children\n");

    /* Make NKIDS children */
    for (ix = 0; ix < NKIDS; ix++) {

        /* begat a child */
        if ((pid = fork()) == -1) {
            printf("fork #1"); exit(1); }

        /* Check to see if child or parent */
        if (pid == 0) {
            /* CHILD */
            child = 1;          /* Set child true */
            parent = 0;         /* and parent false -just to make sure */
            identity    = ix;   /* Set 'local' identity */
            goto CHILD;
        } else {
            /* PARENT */
            parent = 1;         /* I am the parent */
            child = 0;          /* not the child */
        }
    }

    /* PARENT SECTION */
    /* NKIDS children have been created */

    /* Make sure that children are alive */
    usleep(10000);

    for (ix = 0; ix < NKIDS; ix++) {
        /* Read the message queue */
        if((itemp=msgrcv(msgid,&the_word,MAX_MSG_SIZE,ix+1,0)) == -1) {
            perror("Parent >  msgrcv #1 >  "); exit(1); }

        /* Check if the message is from the children */
        if(strncmp("Present",the_word.mtext,itemp) == 0) {
            printf("Child %d Present\n",ix);
        } else {
            printf("Parent >  unexpected message!\n");
        }
    }

    /* Write a message out to each child */
    for (ix = 0; ix < NKIDS; ix++) {

        /* Set the message type -Child id in this case */
        the_word.mtype = ix+ NKIDS;

        /* Set the message text */
        strcpy(the_word.mtext,msgs[ix]);

        /* Send the message */
        if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
            perror("Child >  msgsnd #1 >  "); exit(1); }

        /* Update the message */
        strcpy(msgs[ix],"SENT");

        /* Wait for child to get the message */
        usleep(20000);
    }

    /* Have spoken to everyone, tell all children to quit */
    printf("Parent >  Children quit!\n");

    /* Setup the quit message */
    strcpy(the_word.mtext,"Quit");

    for (ix = 0; ix < NKIDS; ix++) {

        /* Target each child in turn */
        the_word.mtype = ix+ 1+ NKIDS;

        /* Send the message */
        if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
            perror("Parent >  msgsnd #2 >  "); exit(1); }
    }

    /* Wait for children to exit */
    usleep(20000);

    printf("Parent >  All children have quit, completing\n");

    /* Remove the message queue */
    if(msgctl(msgid,IPC_RMID,NULL) == -1) perror("Parent >  msgctl >  ");

    printf("Parent >  Completed\n");
    exit(0);
CHILD:
    /* CHILD */

    /* Tell everyone I am waiting */
    /* Setup the message type */
    the_word.mtype = identity+ 1;

    /* Setup the message text */
    strcpy(the_word.mtext,"Present");

    /* Send the message */
    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
        perror("Child >  msgsnd #3 >  "); exit(1); }

    printf("Child >  #%d waiting\n",identity);

    /* Wait for something to do */
    ix  = 1;
    while(ix) {
        /* Setup the message type first for this child */
        itemp   = identity+ 1+ NKIDS;

        /* Read the message queue */
        if((itemp=msgrcv(msgid,&the_word,MAX_MSG_SIZE,itemp,0)) == -1) {
            perror("Child >  msgrcv #2 >  "); exit(1); }

        /* Am I being told to quit? */
        if(strncmp("Quit",the_word.mtext,itemp) == 0) {
            ix  = 0;    /* Yes, set ix to a 'false' value */
        } else {
            printf("Child %d, received message: %s\n",identity,the_word.mtext);
        }
    }

    /* Have found QUIT, bye bye */
    printf("Child >  #%d quitting\n",identity);

    exit(0);
}
```

The code is compiled and executed in the following way:

```
$ make
   cc -c debug_message.c
   cc -o debug_message debug_message.o
$ debug_message
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child 0 Present
Child 1 Present
Child 2 Present
Child 0, received message: more beer
Child 1, received message:
Segmentation fault (core dumped)
```

We have a segmentation fault. Recompile, using the debug option, re-run the program, and find the most recent core file:

```
$ make
cc -g -c debug_message.c
cc -g -o debug_message debug_message.o
ld: 0711-851 SEVERE ERROR: Output file: debug_message
   The file is in use and cannot be overwritten.
make: *** [debug_message] Error 12
```

There is still a problem:

```
$ ps
PID    TTY  TIME CMD
28660  pts/8  0:00 debug_message
29454  pts/8  0:00 debug_message
29796  pts/8  0:00 ps
31046  pts/8  0:00 sh -is
33904  pts/8  0:00 debug_message
```

debug_message uses fork to create some children and when the parent crashes, the children are left waiting with nothing to do. To make things easier, here is a file script cleanup that uses `awk` to find and kill any instances of debug_message:

```
$ cat cleanup
#!/bin/csh -f
#
# Setup an environment value
setenv TF /tmp/cleaner
# Try to remove the old file -just in case
rm -f $TF
# Create a new empty file
touch $TF
# OK, do a ps and look for 'debug_message'
ps -e | awk '$4 == "debug_message" {print "kill -9 " $1}' >> $TF
# Execute the generated cleaner file
csh $TF
# Cleanup the cleaner
rm $TF
# Not quite sure what this does ;-)
exit
$ cleanup
$ ps
PID    TTY  TIME CMD
29806  pts/8  0:00 ps
31046  pts/8  0:00 sh -is
```

Almost all the zombie children have gone away. Try again:

```
$ make
   cc -g -qcpluscmt -c debug_message.c
   cc -g -qcpluscmt -o debug_message debug_message.o
$ debug_message
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child 0 Present
Child 1 Present
Child 2 Present
Child 0, received message: more beer
Child 1, received message:
Segmentation fault (core dumped)
$ ls -l core*
-rw-r--r--   1 pnutt    staff          8191 Mar 24 13:32 core.23418.24193218
-rw-r--r--   1 pnutt    staff          8191 Mar 24 15:29 core.28452.24212941
```

We then start dbx and see what was happening when it crashed:

```
$ dbx debug_message core.28452.24212941
Type 'help' for help.
reading symbolic information ...
[using memory image in core.28452.24212941]

Segmentation fault in strcpy.strcpy [debug_message] at 0x10000a8c
0x10000a8c (strcpy+0x8c) 94e50004stwu r7,0x4(r5)
(dbx) where
strcpy.strcpy() at 0x10000a8c
main(), line 108 in "debug_message.c"
(dbx) quit
```

There seems to be a problem in strcpy at line 108 in the source file. Start up dbx, run the code until it fails, and investigate:

```
$ dbx debug_message
Type 'help' for help.
reading symbolic information ...
(dbx) cont
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child 0 Present
Child 1 Present
Child 2 Present
Child 0, received message: more beer
Child 1, received message:

Segmentation fault in strcpy.strcpy [debug_message] at 0x10000a8c
0x10000a8c (strcpy+0x8c) 94e50004       stwu   r7,0x4(r5)
(dbx) where
strcpy.strcpy() at 0x10000a8c
main(), line 108 in "debug_message.c"
(dbx) list 108
108    strcpy(msgs[ix],"SENT");
(dbx) whatis msgs
unsigned char * msgs[2];
(dbx) print msgs
("SENT", "SENT")
(dbx) print ix
2
(dbx) quit
```

The code is crashing because `msgs` has been defined with 2 entries and line 108 is trying to update entry number 3 causing `SIGS`EGV. Update the code so it uses the minimum of NKIDS and NUM_OF_MSGS to resolve the problem. This can be done with the magic construct NKIDS>NUM_OF_MSGS?NUM_OF_MSGS:NKIDS, so line 95 will now read:

```
for (ix=0; ix<(NKIDS>NUM_OF_MSGS?NUM_OF_MSGS:NKIDS); ix++) {
```

Recompile and rerun:

```
$ debug_message
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child 0 Present
Child 1 Present
Child 2 Present
Child 0, received message: more beer
Parent >  Children quit!
Child >  #0 quitting
Child >  #1 quitting
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed
```

The code is not running correctly, as only child 0 is printing out a received message. Assuming that messages are working, what is going on? Get `dbx` running again and look at where the missing message is generated (or not): line 104 (see the next screen).

At this line, the message is sent but the message itself is set up on lines 98 and 101 within the structure the_word. dbx is used to list line 104 (list 104) and then examine the type and contents of the structure (whatis the_word, print the_word). As this structure is used throughout the code, it is useful to see where and how it is changed, and this is accomplished by using trace the_word. Just to make sure, we set a breakpoint at line 104 by using stop at 104.

The `cont` command is used to continue execution and we see the_word being updated through the code and the stop at line 104. The trace on the_word shows that message type mtype 3 has already been used and, as this value is used to select individual messages or channels in this case, this may be a problem. In fact, the problem was created by trying to send messages identified by the child identity value, starting at 0. This caused an EINVAL error so all mtype values had to be incremented by 1 to ensure correct handling. If any of the magic value++ points are missed, the messages do not get to the right place. In this instance, `dbx` is used to add 1 to the mtype value and the code permitted to run to completion:

```
$ dbx debug_message
Type 'help' for help.
reading symbolic information ...
(dbx) list 104
104    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
(dbx) whatis the_word
struct mess the_word;
(dbx) print the_word
(mtype = 0, mtext = "")
(dbx) trace the_word
[1] trace the_word
(dbx) stop at 104
[2] stop at 104
(dbx) cont
initially (at line 37 in "debug_message.c"):the_word = (mtype = 0, mtext = "")
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
after line 83 in "debug_message.c":the_word = (mtype = 1, mtext = "Present")
Child 0 Present
after line 83 in "debug_message.c":the_word = (mtype = 2, mtext = "Present")
Child 1 Present
after line 83 in "debug_message.c":the_word = (mtype = 3, mtext = "Present")
Child 2 Present
after line 101 in "debug_message.c":the_word = (mtype = 3, mtext = "cd /pub")
[2] stopped in main at line 104
104    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
(dbx) print the_word
(mtype = 3, mtext = "cd /pub")
(dbx) assign the_word.mtype=4
(dbx) cont
Child 0, received message: cd /pub
after line 104 in "debug_message.c":the_word = (mtype = 4, mtext = "cd /pub")
after line 101 in "debug_message.c":the_word = (mtype = 4, mtext = "more beer")
[2] stopped in main at line 104
104    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
(dbx) print the_word
(mtype = 4, mtext = "more beer")
(dbx) assign the_word.mtype=the_word.mtype+1
(dbx) cont
Child 1, received message: more beer
after line 104 in "debug_message.c":the_word = (mtype = 5, mtext = "more beer")
Parent >  Children quit!
after line 118 in "debug_message.c":the_word = (mtype = 5, mtext = "Quit")
after line 123 in "debug_message.c":the_word = (mtype = 4, mtext = "Quit")
Child >  #0 quitting
after line 123 in "debug_message.c":the_word = (mtype = 5, mtext = "Quit")
Child >  #1 quitting
after line 123 in "debug_message.c":the_word = (mtype = 6, mtext = "Quit")
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed
execution completed
(dbx) quit
```

When run, the output now reads:

```
pnutt@aix510 5=debug_message
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child 0 Present
Child 1 Present
Child 2 Present
Child 0, received message: cd /pub
Child 1, received message: more beer
Parent >  Children quit!
Child >  #0 quitting
Child >  #1 quitting
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed
```

The code is now running correctly.

12.2.1 Endianness and 32-bit/64-bit problem

The file `debug_message.c` has been configured to run on a little endian/ILP32 system. We need to get it running on a big endian machine and use the LP64 model. Here is the code:

```
/*
* Program debug_message.C
*
* This program has been modified (yet again) to exhibit a few bugs that can be
* investigated with the debugger.
*
* The program generates a number of child processes using fork and
* communicates between them using messages.
*
*/
#include <iostream.h>   /* Needed for printf */
#include <sys/types.h>  /* Needed for fork */
#include <unistd.h>     /* Needed for fork */
#include <sys/msg.h>    /* Needed for msgget... */
#include <string.h>     /* Needed for strncmp */
#include <errno.h>      /* Needed for perror */
#include <stdio.h>      /* Needed for perror */
#include <stdlib.h>     /* Needed for exit */

/* maximum message size */
#define MAX_MSG_SIZE    64

#define NKIDS   3               /* How many children? */
#define NUM_OF_MSGS     2       /* How many predefined messages are there? */

main(void)
{
    /* Setup the union that allows an long val in a char record */
    union mtext_u_t {
```

```
    char    mtext[MAX_MSG_SIZE];
    long    mval[MAX_MSG_SIZE/4];
};

/* Setup the message structure */
struct mess {
    long            mess_type;
    mtext_u_t       mtext_u;
} the_word;

/* Setup the special messages */
char    msgs[NUM_OF_MSGS][MAX_MSG_SIZE] = { "cd /pub","more beer"};

int identity    = -1;           /* Define and initialize */
int ix  = -1;                   /* Define and initialize */
pid_t   pid = -1;               /* Define and initialize */
int     nkids = -1;             /* Define and initialize */
int     child = 0;              /* Set child to false */
int     parent = 0;             /* Set parent to false */
int     itemp   = -1;
int     msgid   = -1;

/* Initialise a message queue */
itemp   = IPC_CREAT | 0777;
if((msgid=msgget(IPC_PRIVATE,itemp))==-1) {
    perror("Parent >  msgget #1 >  "); exit(1); }

printf("Parent >  Making children\n");

/* Make NKIDS children */
for (ix = 0; ix < NKIDS; ix++) {

    /* begat a child */
    if ((pid = fork()) == -1) {
        perror("Parent >  fork #2 >  "); exit(1); }

    /* Check to see if child or parent */
    if (pid == 0) {
        /* CHILD */
        child = 1;      /* Set child true */
        parent = 0;     /* and parent false -just to make sure */
        identity        = ix;   /* Set 'local' identity */
        goto CHILD;
    } else {
        /* PARENT */
        parent = 1;     /* I am the parent */
        child = 0;      /* not the child */
    }
}

/* PARENT SECTION */
/* NKIDS children have been created */

/* Make sure that children are alive */
usleep(10000);

for (ix = 0; ix < NKIDS; ix++) {
    /* Read the message queue */
    if((itemp=msgrcv(msgid,&the_word,MAX_MSG_SIZE,ix+1,0)) == -1) {
        perror("Parent >  msgrcv #3 >  "); exit(1); }

    /* Check if the message is from the children */
    if(strncmp("Present",the_word.mtext_u.mtext,itemp) == 0) {
        printf("Child >  #%d Present\n",ix);
    } else {
        printf("Parent >  unexpected message!\n");
    }
}

/* Write a message out to each child */
for (ix = 0; ix < NKIDS; ix++) {

    /* Set the message type -Child id in this case */
    the_word.mess_type = ix+ NKIDS;

    /* Set the message text */
    strcpy(the_word.mtext_u.mtext,msgs[ix]);

    /* Send the message */
    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
        perror("Parent >  msgsnd #4 >  "); exit(1); }

    /* Wait for child to get the message */
    usleep(20000);
}

/* Send a special message to child 1 */
the_word.mess_type      = NKIDS+ 2;
strcpy(the_word.mtext_u.mtext,"Special");

/* Setup the control word */
the_word.mtext_u.mtext[60]      = (char)0xde;
the_word.mtext_u.mtext[61]      = (char)0xfa;
the_word.mtext_u.mtext[62]      = (char)0xed;
the_word.mtext_u.mtext[63]      = (char)0xfe;

/* Send the message */
if(msgsnd(msgid,&the_word,sizeof(the_word.mtext_u),0) != 0) {
    perror("Parent >  msgsnd #5 >  "); exit(1); }

/* Wait for child */
usleep(20000);

/* Have spoken to everyone, tell all children to quit */
printf("Parent >  Children quit!\n");

/* Setup the quit message */
strcpy(the_word.mtext_u.mtext,"Quit");

for (ix = 0; ix < NKIDS; ix++) {

    /* Target each child in turn */
    the_word.mess_type = ix+ 1+ NKIDS;

    /* Send the message */
    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
        perror("Parent >  msgsnd #6 >  "); exit(1); }
}

/* Wait for children to exit */
usleep(20000);

printf("Parent >  All children have quit, completing\n");

/* Remove the message queue */
if(msgctl(msgid,IPC_RMID,NULL) == -1) perror("Parent >  msgctl >  ");
```

```
    printf("Parent >  Completed\n");

    exit(0);
CHILD:
    /* CHILD */

    /* Tell everyone I am waiting */
    /* Setup the message type */
    the_word.mess_type = identity+ 1;

    /* Setup the message text */
    strcpy(the_word.mtext_u.mtext,"Present");

    /* Send the message */
    if(msgsnd(msgid,&the_word,MAX_MSG_SIZE,0) != 0) {
        perror("Child >  msgsnd #7 >  "); exit(1); }

    printf("Child >  #%d waiting\n",identity);

    /* Wait for something to do */
    ix      = 1;
    while(ix) {
        /* Setup the message type first for this child */
        itemp   = identity+ 1+ NKIDS;

        /* Read the message queue */
        if((itemp=msgrcv(msgid,&the_word,MAX_MSG_SIZE,itemp,0)) == -1) {
            perror("Child >  msgrcv #8 >  "); exit(1); }

        /* Am I being told to quit? */
        if(strncmp("Quit",the_word.mtext_u.mtext,itemp) == 0) {
            ix      = 0;    /* Yes, set ix to a 'false' value */
        } else if(strncmp("Special",the_word.mtext_u.mtext,itemp) == 0) {
            printf("Child #%d, Special message received: %x\n",
                identity,the_word.mtext_u.mval[15]);
        } else {
            printf("Child >  #%d received message %s\n",
                identity,the_word.mtext_u.mtext);
        }
    }

    /* Have found QUIT, bye bye */
    printf("Child >  #%d quitting\n",identity);

    exit(0);
}
```

Compile the code using -g, just in case we have problems and generate two executables, one for ILP32 mode and one for LP64 mode:

```
pnutt@aix510 1=make
xlc -g -c debug_message.c
mv debug_message.o debug_message32.o
xlc -g -o debug_message32 debug_message32.o
xlc -g -q64 -c debug_message.c
mv debug_message.o debug_message64.o
xlc -g -q64 -o debug_message64 debug_message64.o
pnutt@aix510 2=debug_message32
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message more beer
Child >  #1 received message ÿÿÿÿ
Child #1, Special message received: defaedfe
Parent >  Children quit!
Child >  #0 quitting
Child >  #1 quitting
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed
```

The code has been run in 32-bit mode and runs to completion, but the second child message is missing again and the special message text definitely looks wrong. Duplicate the changes from the original debug_message.c code, recompile, and start up the debugger.

The following output shows the `dbx` session. This time, we set breakpoints at lines 121 and 127 to look at the word setup. Being suspicious of a char/long union with explicit byte addressing, we get `dbx` to run a standard subroutine swab, which switches adjacent bytes within the special message word. The data is printed out in hexadecimal and looks a lot better, but the words are still incorrect. As the code runs correctly on the original little endian platform and we are running on a big endian platform, this appears to be an endianness problem.

```
pnutt@aix510 1=dbx debug_message32
Type 'help' for help.
reading symbolic information ...
(dbx) l 120
120                 /* Setup the control word */
(dbx) l
121                 the_word.mtext_u.mtext[60]= (char)0xde;
122                 the_word.mtext_u.mtext[61]= (char)0xfa;
123                 the_word.mtext_u.mtext[62]= (char)0xed;
124                 the_word.mtext_u.mtext[63]= (char)0xfe;
125
126                 /* Send the message */
127                 if(msgsnd(msgid,&the_word,sizeof(the_word.mtext_u),0) != 0) {
128                     perror("Parent >  msgsnd #5 >  "); exit(1); }
129
130                 /* Wait for child */
(dbx) stop at 121
[1] stop at "debug_message.C":121
(dbx) stop at 127
[2] stop at "debug_message.C":127
(dbx) c
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message cd /pub
Child >  #1 received message more beer
[1] stopped in main at line 121 in file "debug_message.C"
121                 the_word.mtext_u.mtext[60]= (char)0xde;
(dbx) s
stopped in main at line 122 in file "debug_message.C"
122                 the_word.mtext_u.mtext[61]= (char)0xfa;
(dbx) s
stopped in main at line 123 in file "debug_message.C"
123                 the_word.mtext_u.mtext[62]= (char)0xed;
(dbx) s
stopped in main at line 124 in file "debug_message.C"
124                 the_word.mtext_u.mtext[63]= (char)0xfe;
(dbx) s
[2] stopped in main at line 127 in file "debug_message.C"
127                 if(msgsnd(msgid,&the_word,sizeof(the_word.mtext_u),0) != 0) {
(dbx) call swab(&the_word.mtext_u.mval[15],&the_word.mtext_u.mval[15],4)

swab returns successfully
(dbx) p the_word.mtext_u.mval[15]
0xfadefeed
(dbx) c
Child #1, Special message received: fadefeed
Parent >  Children quit!
```

Chapter 2, “Endianness - byte ordering” on page 9 has some programming examples. For example, the example in Figure 7 on page 15 is used to detect the endianness of the run time system and take the appropriate action. The modified sections of code are shown below:

```
.
.
.
#define NUM_OF_MSGS 2   /* How many predefined messages are there? */

/* Check if we are big or little endian */
inline int is_bigendian()
{
    const int endian = 1;     /* Setup the LSB */
    return(1-*(char*)&endian);    /* Check and invert boolean */
}
.
.
.
/* Send a special message to child 1 */
    the_word.mess_type  = NKIDS+ 2;
    strcpy(the_word.mtext_u.mtext,"Special");

    if(is_bigendian()) {
        /* Setup the control word -written for a LE machine */
        printf("BIG Endian system\n");
        the_word.mtext_u.mtext[63]  = (char)0xde;
        the_word.mtext_u.mtext[62]  = (char)0xfa;
        the_word.mtext_u.mtext[61]  = (char)0xed;
        the_word.mtext_u.mtext[60]  = (char)0xfe;
    } else {
        /* Setup the control word -written for a BE machine */
        printf("LITTLE Endian system\n");
        the_word.mtext_u.mtext[60]  = (char)0xde;
        the_word.mtext_u.mtext[61]  = (char)0xfa;
        the_word.mtext_u.mtext[62]  = (char)0xed;
        the_word.mtext_u.mtext[63]  = (char)0xfe;
    }

    /* Send the message */
    if(msgsnd(msgid,&the_word,sizeof(the_word.mtext_u),0) != 0) {
        perror("Parent >  msgsnd #5 >  "); exit(1); }
.
.
.
```

The code has been updated, so we will try to run it in 32-bit mode and then again in 64-bit mode:

```
pnutt@aix510 1=debug_message32
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message cd /pub
Child >  #1 received message more beer
BIG Endian system
Child #1, Special message received: feedfade
Parent >  Children quit!
Child >  #0 quitting
Child >  #1 quitting
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed

pnutt@aix510 2=debug_message64
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message cd /pub
Child >  #1 received message more beer
BIG Endian system
Child >  msgrcv #8 >  : Arg list too long
Parent >  Children quit!
Child >  #0 quitting
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed
```

The 32-bit code runs correctly, but the 64-bit version has a problem reported from msgrcv. Run dbx and look at the msgsnd call that is sending the message to check out the argument list:

```
pnutt@aix510 1=dbx debug_message64
Type 'help' for help.
reading symbolic information ...

(dbx) stop in main
[1] stop in main
(dbx) c
[1] stopped in main at line 48 in file "debug_message.C"
   48       char    msgs[NUM_OF_MSGS][MAX_MSG_SIZE] = { "cd /pub","more beer"};
(dbx) stop at 145
[3] stop at "debug_message.C":145
(dbx) c
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message cd /pub
Child >  #1 received message more beer
BIG Endian system
[3] stopped in main at line 145 in file "debug_message.C"
  145       if(msgsnd(msgid,&the_word,sizeof(the_word.mtext_u),0) != 0) {
(dbx) dump
main(), line 145 in "debug_message.C"
pid = 25892
msgid = 1966155
itemp = 64
parent = 1
child = 0
nkids = -1
ix = 2
identity = -1
msgs = (
"cd /pub"
"more beer"
)
the_word = (mess_type = 5, mtext_u = [union])
(dbx) print sizeof(the_word.mtext_u)
128
(dbx) quit
```

msgsnd is sending a message of 128 bytes, MAX_MSG_SIZE is defined at 64. This could be a ILP32/LP64 model issue, but the easiest fix is to change the msgsnd call to use MAX_MSG_SIZE. Modify the code, recompile, and run it again:

```
pnutt@aix510 1=debug_message64
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message cd /pub
Child >  #1 received message more beer
BIG Endian system
Child #1, Special message received: ffffff80
Parent >  Children quit!
Child >  #0 quitting
Child >  #1 quitting
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed
```

The `msgsnd` error has gone away but the special message text is wrong again. We will run `dbx` again, but this time, we will set multproc on so we can debug the child process where the special message is output. The output from the two `dbx` sessions is shown in the next screen. The first block is from the parent `dbx` session and the second block is the child `dbx` session. Both sessions ran concurrently and allowed `dbx` interaction with the both running processes:

```
pnutt@aix510 1=dbx debug_message64
(dbx) multproc on
(dbx) c
Parent >  Making children
application forked, child pid=26208, process stopped, awaiting input

stopped due to fork with multiprocessing enabled in . at 0x377c
0x000000000000377c e9a1ff68         ld   r13,-152(r1)
(dbx) c
Child >  #0 waiting
(dbx) c
application forked, child pid=21900, process stopped, awaiting input

stopped due to fork with multiprocessing enabled in . at 0x377c
0x000000000000377c e9a1ff68         ld   r13,-152(r1)
(dbx) c
Child >  #1 waiting
application forked, child pid=30196, process stopped, awaiting input

stopped due to fork with multiprocessing enabled in . at 0x377c
0x000000000000377c e9a1ff68         ld   r13,-152(r1)
(dbx) c
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message cd /pub
Child >  #1 received message more beer
BIG Endian system
Parent >  Children quit!
Child >  #0 quitting
Child >  #2 quitting
Parent >  All children have quit, completing
Parent >  Completed

execution completed

(dbx) q
```

When a fork occurs, another debug session is started, and this allows handshaking and IPC to be investigated, if necessary. The following shows the debug session attached to the relevant child process:

```
debugging child, pid=21900, process stopped, waiting input

stopped due to fork with multiprocessing enabled in . at 0x377c
0x000000000000377c e9a1ff68        ld   r13,-152(r1)
(dbx) stop at 207
[3] stop at "debug_message.C":207
(dbx) c

Trace/BPT trap in . at 0x377c
0x000000000000377c e9a1ff68        ld   r13,-152(r1)
(dbx) c
[3] stopped in main at line 207 in file "debug_message.C"
  207      printf("Child #%d, Special message received:
%x\n",identity,the_word.mtext_u.mval[15]);
(dbx) whatis the_word.mtext_u.mval
long mval[16];
(dbx) set $hexints
(dbx) print  the_word.mtext_u.mval
(0x5370656369616c00, 0x72001000a01f2ed8, 0x8, 0x64, 0x9001000a01daa78, 0x1, 0x64,
0x1feedfade,
0x0, 0x80000000, 0xfffffffffffffe90, 0xffffffffffffff80, 0xbadc0ffee0ddf00d,
0x1000004bc,
0x800200140030000, 0xffffffffffffff80)
(dbx) set $pretty="on"
(dbx) print  the_word.mtext_u.mval
[0] = 0x5370656369616c00
[1] = 0x72001000a01f2ed8
[2] = 0x8
[3] = 0x64
[4] = 0x9001000a01daa78
[5] = 0x1
[6] = 0x64
[7] = 0x1feedfade
[8] = 0x0
[9] = 0x80000000
[10] = 0xfffffffffffffe90
[11] = 0xffffffffffffff80
[12] = 0xbadc0ffee0ddf00d
[13] = 0x1000004bc
[14] = 0x800200140030000
[15] = 0xffffffffffffff80
(dbx) unset $hexints
```

A breakpoint is set where the special message is output (line 207). The contents of the_word.mtext_u.mval is printed. A hexadecimal output format is requested with the `set $hexints` command to `dbx`. The output is not clear, so we use `set $pretty="on"`. This time, the output looks a bit easier to read. mval[15] is definitely 0xffffff80, but mval[7] contains some of the data we want 0xfeedfade. When we look at sizeof(long), it shows that it is eight bytes, but looking at the code, line 31 is long mval[MAX_MSG_SIZE/4];. This creates two problems:

- First, if it really needs four bytes per word, the code should use int and then use sizeof(int) in the divide rather than 4.

- As the code uses hardcoded byte addresses for the special message word, which is in an int/char union, it has to use four bytes per word. Modify the code, recompile, and try again:

```
pnutt@aix510 1=debug_message64
Parent >  Making children
Child >  #0 waiting
Child >  #1 waiting
Child >  #2 waiting
Child >  #0 Present
Child >  #1 Present
Child >  #2 Present
Child >  #0 received message cd /pub
Child >  #1 received message more beer
BIG Endian system
Child #1, Special message received: feedfade
Parent >  Children quit!
Child >  #0 quitting
Child >  #2 quitting
Child >  #1 quitting
Parent >  All children have quit, completing
Parent >  Completed
```

The code succeeds.

12.3 idebug

idebug (IBM Distributed Debugger) is a graphical debugger. Depending on the needs, it is possible to run the debugger directly from a graphical terminal connected to the host running the code, or run from a client on a personal work station connecting to a remote server. The idebug system is supplied with the C for AIX Version 5 and VisualAge C++ Professional for AIX Version 5 compilers.

The next example is a piece of code which tries to compute a Mandelbrot set:

```
#include <pthread.h>
#include <stdlib.h>
#include <errno.h>

/***************************************/
/* Values may be changed               */
/***************************************/
#define MAX_ITERATION 1024
#define MAX_LENGTH 100
#define X_MIN -2.1
#define Y_MIN -1.1
#define X_MAX 0.7
#define Y_MAX 1.1
#define RESOLUTION 20 /* vertical resolution, horizontal is then derived */

/****************************************/
```

```
/* Do not change the following variables */
/*****************************************/
#define COLORS 11

typedef struct {
  int y, startx, endx;
} interval;

char *col = " -:=+oxOX@#";
int **pixels;
int xres = RESOLUTION*3.2;
int yres = RESOLUTION;
float xmin = X_MIN, ymin = Y_MIN;
float xstep = (X_MAX-X_MIN)/(RESOLUTION*3.2);
float ystep = (Y_MAX-Y_MIN)/RESOLUTION;

/************************************************/
/* Compute row, subinterval specified in *argy   */
/* if computation takes more than 3 seconds      */
/* split into two threads                        */
/************************************************/
void *row(void *argy)
{
  pthread_t th1, th2;
  time_t start, now;
  int x, y ,iteration = 0, cindx = 0;
  int rc;
  interval *intv, int1, int2;
  float z1 = 0.0, z2 = 0.0, t1;

  intv = (interval *)argy;

  /* record start time of this thread */
  start = time(NULL);

  for(x = intv->startx; x <= intv->endx; x++, cindx = 0, iteration = 0, z1 = 0.0, z2 =
0.0) {
    /* compute one pixel */
    do {
      t1 = z1*z1-z2*z2+(xmin+x*xstep);
      z2 = 2*z1*z2+(ymin+intv->y*ystep);
      z1 = t1;
      iteration++;
    }
    while(iteration < MAX_ITERATION && z1*z1+z2*z2 < MAX_LENGTH);

    do
      cindx++;
    while((1 << cindx) < iteration);

    pixels[intv->y][x] = (int)((iteration >= MAX_ITERATION) ? 0 : (cindx%COLORS));

    /* record time passed in this thread */
    now = time(NULL);

    /* if too much time has elpased, start two new threads */
    if (difftime(now,start) > 3.0 && intv->endx-x > 3) {
      /* printf("Splitting row %d: (%d,%d)\n", intv->y, x+1, intv->endx); */
      int1.y = int2.y = intv->y;
      int1.startx = x++;
      int1.endx = (intv->endx + x)/2;
      int2.startx = (intv->endx + x)/2+1;
      int2.endx = intv->endx;
      rc = pthread_create(&th1, NULL, row, (void *)&int1);
      if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
    exit(1);

      rc = pthread_create(&th2, NULL, row, (void *)&int2);
      if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
    exit(1);

      rc = pthread_join(th1, NULL);
      if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
    exit(1);

      rc = pthread_join(th2, NULL);
      if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
    exit(1);

      x = intv->endx;
    }
  }
  pthread_exit(0);
}

int main(int argc, char **argv)
{
  pthread_t mythreads[RESOLUTION];
  interval intv;
  int x,y;
  int rc;

  /* allocate memory for fractal pixels */
  if(NULL == (pixels = (int **)malloc(sizeof(int *)*(long)yres))) {
    perror("malloc");
    exit(0);
  }

  for(y = 0; y < yres; y++)
    if(NULL == (pixels[y] = (int *)malloc(sizeof(int)*(long)xres))) {
      perror("malloc");
      exit(0);
    }

  /* start all row-threads */
  for(y = 0; y < yres; y++) {
    intv.y = y;
    intv.startx = 0;
    intv.endx = xres-1;

    rc = pthread_create(&mythreads[y], NULL, row, (void *)&intv);
    if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
      exit(1);
  }

  /* join all threads */
  for(y = 0; y < yres; y++) {
    rc = pthread_join(mythreads[y], NULL);
    if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
      exit(1);
  }

  /* print out fractal pixels */
  for(y = 0;y < yres; y++) {
    for(x = 0; x < xres; x++)
```

```
      putchar(col[pixels[y][x]]);
    putchar('\n');
  }

  /* free allocated memory */
  for(y = 0; y < yres; y++)
    free(pixels[y]);
  free(pixels);

  exit(0);
}
```

The program is compiled:

```
cc_r -o debugmandelbrot debugmandelbrot.c
```

and run. It gives unexpected results, as shown below. Lines, which seem to be part of the expected result, are printed randomly:

```
$ debugmandelbrot







::::===============+++++++++++++o                         x=====








:::::::::::::=====================+oo++oxx0o0  oo#+0++++o+======


::::::::::::::::::::::::::==================+o+o+==========:::::
$
```

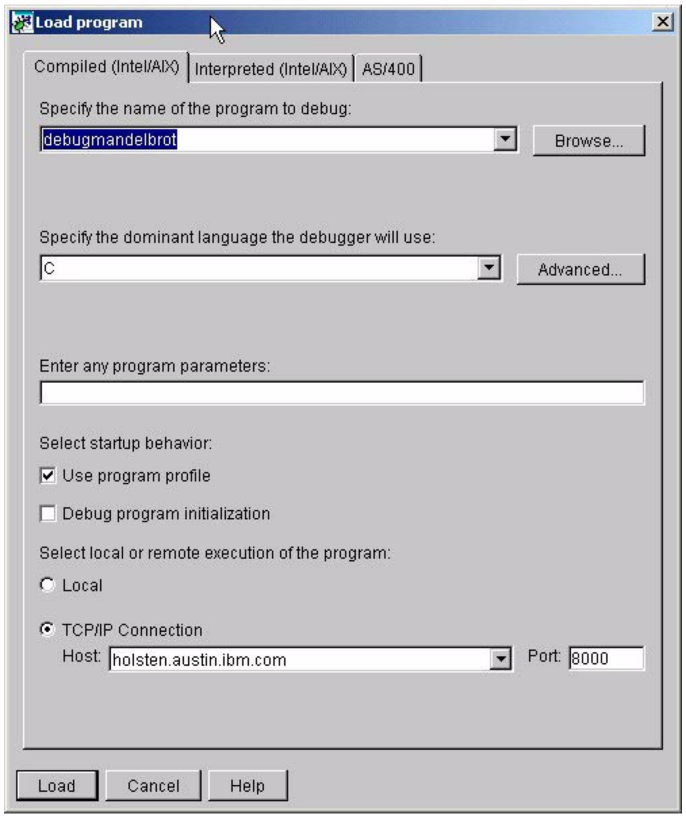
We choose to recompile for debugging with:

```
cc_r -g -o debugmandelbrot debugmandelbrot.c
```

and start the idebug program. We will try to debug remotely, so we start the program irmtdbgc on the remote host:

```
$ pwd
/tmp
$ ls debugmandelbrot*
debugmandelbrot debugmandelbrot.c
$ irmtdbgc
IBM Distributed Debugger
Version 8.5 (10/6/00) - Licenced Material - Property of IBM
(c) Copyright IBM Corp 1991, 2000 - All Rights Reserved
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADB Schedule Contract with IBM Corp.

Initializing communication: protocol=tcpip port=8000 connect=wait
Waiting for connection...
```

On the client machine, we start idebug and log on, choosing the debugthread program, and pressing the load button (see Figure 71).


Load program

Compiled (Intel/AIX) | Interpreted (Intel/AIX) | AS/400

Specify the name of the program to debug: debugmandelbrot

Specify the dominant language the debugger will use: C

Enter any program parameters:

Select startup behavior:

[✓] Use program profile

[ ] Debug program initialization

Select local or remote execution of the program:

( ) Local

(●) TCP/IP Connection

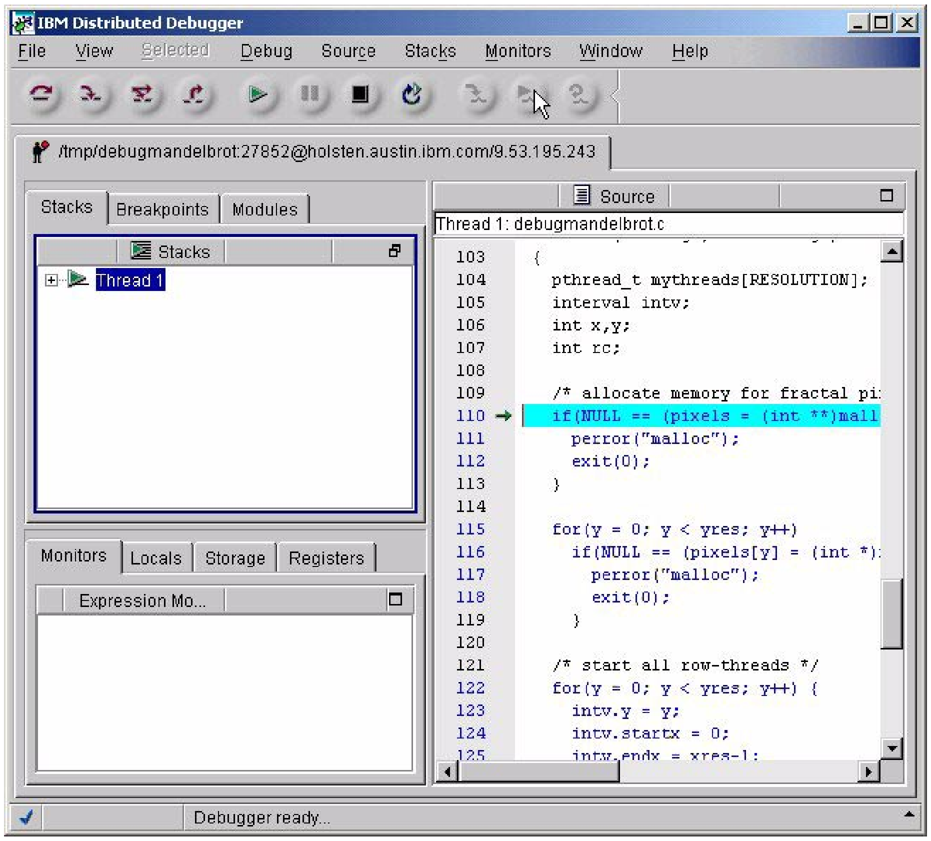
Host: holsten.austin.ibm.com Port: 8000

Figure 71. Selecting the program to debug

On the server, irmtdbc will confirm that a connection has been established:

```
$ pwd
/tmp
$ ls debugmandelbrot*
debugmandelbrot debugmandelbrot.c
$ irmtdbgc
IBM Distributed Debugger
Version 8.5 (10/6/00) - Licenced Material - Property of IBM
(c) Copyright IBM Corp 1991, 2000 - All Rights Reserved
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADB Schedule Contract with IBM Corp.

Initializing communication: protocol=tcpip port=8000 connect=wait
Waiting for connection...
Connection established
```

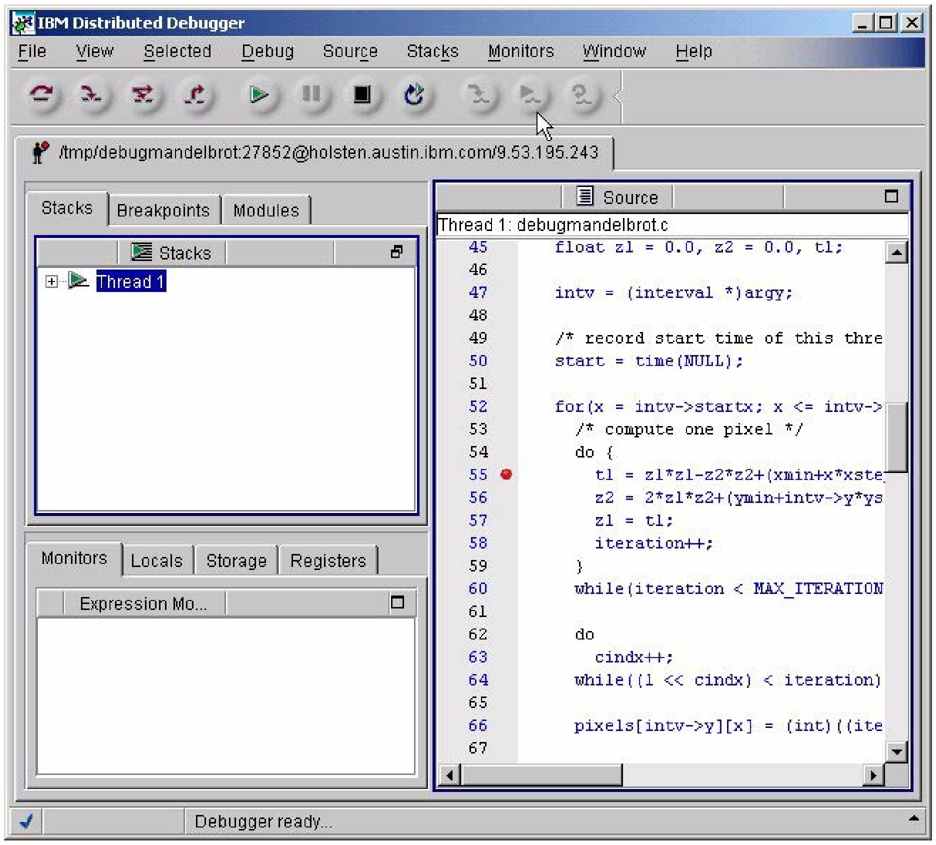
You will see the panel shown in Figure 72 on page 442.


IBM Distributed Debugger
File View Selected Debug Source Stacks Monitors Window Help
/tmp/debugmandelbrot:27852@holsten.austin.ibm.com/9.53.195.243
Stacks Breakpoints Modules
Stacks
Thread 1
Monitors Locals Storage Registers
Expression Mo...
Source
Thread 1: debugmandelbrot.c
Debugger ready...

```
{
  pthread_t mythreads[RESOLUTION];
  interval intv;
  int x,y;
  int rc;

  /* allocate memory for fractal pi
  if(NULL == (pixels = (int **)mall
    perror("malloc");
    exit(0);
  }

  for(y = 0; y < yres; y++)
    if(NULL == (pixels[y] = (int *):
      perror("malloc");
      exit(0);
    }

  /* start all row-threads */
  for(y = 0; y < yres; y++) {
    intv.y = y;
    intv.startx = 0;
    intv.endx = xres-1;
```

Figure 72. Distributed debugger main panel

We start by setting a breakpoint in the computation of the complex number in line 55, as shown in Figure 73 on page 443.

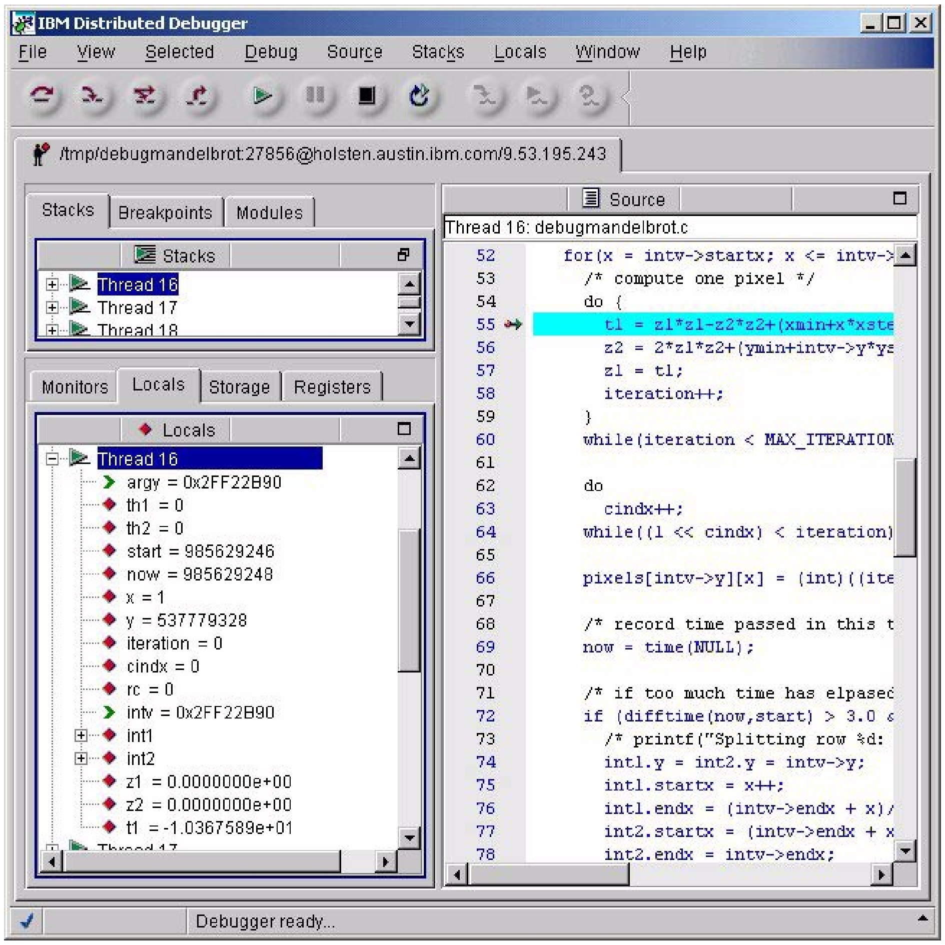

IBM Distributed Debugger
File View Selected Debug Source Stacks Monitors Window Help
/tmp/debugmandelbrot:27852@holsten.austin.ibm.com/9.53.195.243
Stacks Breakpoints Modules
Stacks
Thread 1
Monitors Locals Storage Registers
Expression Mo...
Source
Debugger ready...

Thread 1: debugmandelbrot.c

```
45      float z1 = 0.0, z2 = 0.0, t1;
46
47      intv = (interval *)argv;
48
49      /* record start time of this thre
50      start = time(NULL);
51
52      for(x = intv->startx; x <= intv->
53        /* compute one pixel */
54        do {
55 ●        t1 = z1*z1-z2*z2+(xmin+x*xste
56          z2 = 2*z1*z2+(ymin+intv->y*ys
57          z1 = t1;
58          iteration++;
59        }
60        while(iteration < MAX_ITERATION
61
62        do
63          cindx++;
64        while((1 << cindx) < iteration)
65
66        pixels[intv->y][x] = (int)((ite
67
```

Figure 73. Breakpoint at line 55

We press the play button (it is the fifth round button from the left on the top panel shown in Figure 73), which will run the application until a break point is reached. At this point, the upper left window with the tab name Stacks will list the threads which have been created, as shown in Figure 74 on page 444.


IBM Distributed Debugger
File View Selected Debug Source Stacks Locals Window Help
Stacks Breakpoints Modules
Stacks
Monitors Locals Storage Registers
Locals
Source
Debugger ready...

/tmp/debugmandelbrot:27856@holsten.austin.ibm.com/9.53.195.243

```
Thread 16
Thread 17
Thread 18
```

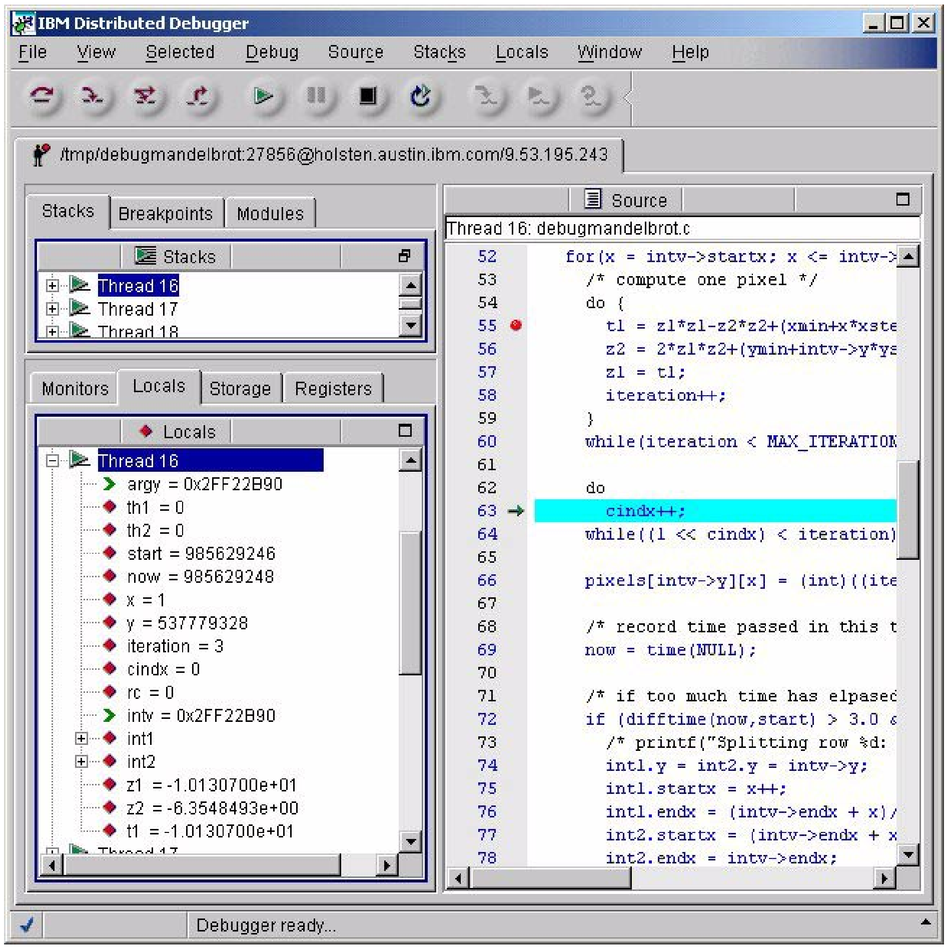
```
Thread 16
    argy = 0x2FF22B90
    th1 = 0
    th2 = 0
    start = 985629246
    now = 985629248
    x = 1
    y = 537779328
    iteration = 0
    cindx = 0
    rc = 0
    intv = 0x2FF22B90
    int1
    int2
    z1 = 0.0000000e+00
    z2 = 0.0000000e+00
    t1 = -1.0367589e+01
Thread 17
```

Thread 16: debugmandelbrot.c

```
for(x = intv->startx; x <= intv->
  /* compute one pixel */
  do {
    t1 = z1*z1-z2*z2+(xmin+x*xste
    z2 = 2*z1*z2+(ymin+intv->y*ys
    z1 = t1;
    iteration++;
  }
  while(iteration < MAX_ITERATION

  do
    cindx++;
  while((1 << cindx) < iteration)

  pixels[intv->y][x] = (int)((ite

  /* record time passed in this t
  now = time(NULL);

  /* if too much time has elpased
  if (difftime(now,start) > 3.0 &
    /* printf("Splitting row %d:
    int1.y = int2.y = intv->y;
    int1.startx = x++;
    int1.endx = (intv->endx + x)/
    int2.startx = (intv->endx + x
    int2.endx = intv->endx;
```

Figure 74. Variables in thread 16

We press the play button a few times, following the threads which happen to be scheduled, and observe that the computation of the complex number (variables z1 and z2) is indeed taking place. We can step into the current source code line and follow the computation one line at a time, using the second round button from the left, as shown in Figure 75 on page 445.


IBM Distributed Debugger
File View Selected Debug Source Stacks Locals Window Help
/tmp/debugmandelbrot:27856@holsten.austin.ibm.com/9.53.195.243
Stacks Breakpoints Modules
Stacks
Thread 16
Thread 17
Thread 18
Monitors Locals Storage Registers
Locals
Source
Debugger ready...

```
Thread 16
  argy = 0x2FF22B90
  th1 = 0
  th2 = 0
  start = 985629246
  now = 985629248
  x = 1
  y = 537779328
  iteration = 3
  cindx = 0
  rc = 0
  intv = 0x2FF22B90
  int1
  int2
  z1 = -1.0130700e+01
  z2 = -6.3548493e+00
  t1 = -1.0130700e+01
Thread 17
```

Thread 16: debugmandelbrot.c

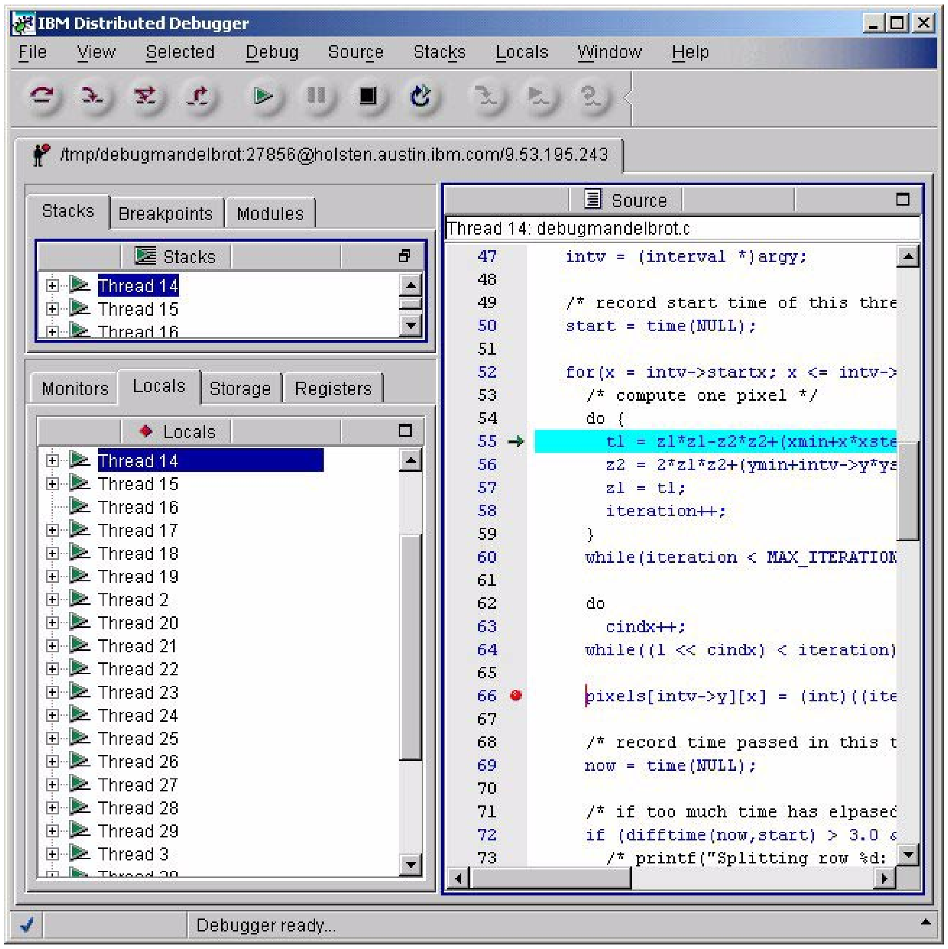
```
52      for(x = intv->startx; x <= intv->
53        /* compute one pixel */
54        do {
55          t1 = z1*z1-z2*z2+(xmin+x*xste
56          z2 = 2*z1*z2+(ymin+intv->y*ys
57          z1 = t1;
58          iteration++;
59        }
60        while(iteration < MAX_ITERATION
61
62        do
63          cindx++;
64        while((1 << cindx) < iteration)
65
66        pixels[intv->y][x] = (int)((ite
67
68        /* record time passed in this t
69        now = time(NULL);
70
71        /* if too much time has elpased
72        if (difftime(now,start) > 3.0 &
73          /* printf("Splitting row %d:
74          int1.y = int2.y = intv->y;
75          int1.startx = x++;
76          int1.endx = (intv->endx + x)/
77          int2.startx = (intv->endx + x
78          int2.endx = intv->endx;
```

Figure 75. Stepping through the code one line at a time

We now believe that the problem lies elsewhere. The code printing out the results of the computations (line 140 - 144) looks correct:

```
for(y = 0;y < yres; y++) {
    for(x = 0; x < xres; x++)
      putchar(col[pixels[y][x]]);
    putchar('\n');
  }
```

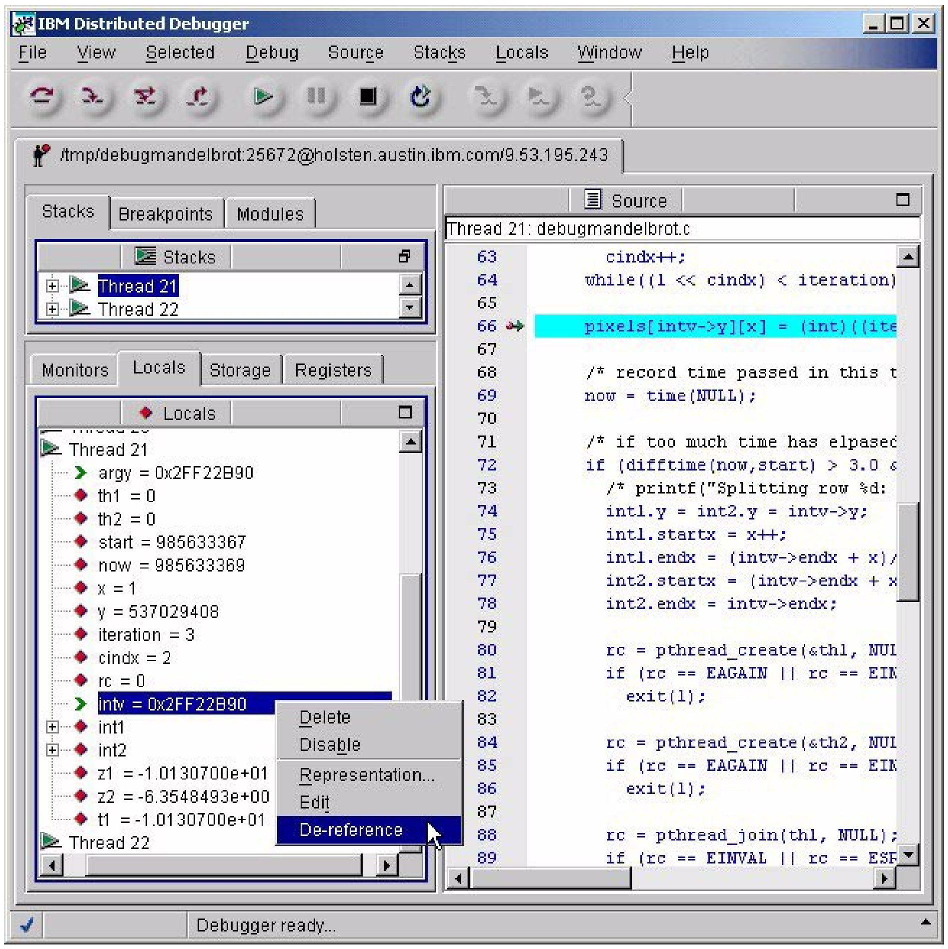
so we remove the break point at line 55 and create a new one at line 66, as shown in Figure 76 on page 446.



```
47      intv = (interval *)argy;
48
49      /* record start time of this thre
50      start = time(NULL);
51
52      for(x = intv->startx; x <= intv->
53        /* compute one pixel */
54        do {
55          t1 = z1*z1-z2*z2+(xmin+x*xste
56          z2 = 2*z1*z2+(ymin+intv->y*ys
57          z1 = t1;
58          iteration++;
59        }
60        while(iteration < MAX_ITERATION
61
62        do
63          cindx++;
64        while((1 << cindx) < iteration)
65
66        pixels[intv->y][x] = (int)((ite
67
68        /* record time passed in this t
69        now = time(NULL);
70
71        /* if too much time has elpased
72        if (difftime(now,start) > 3.0 &
73          /* printf("Splitting row %d:
```

Figure 76. New breakpoint at line 66

We press the play button. At that time, we would like to see what pixels are being updated. We look up the values involved; for that, we need to de-reference the intv variable, as shown in Figure 77 on page 447.

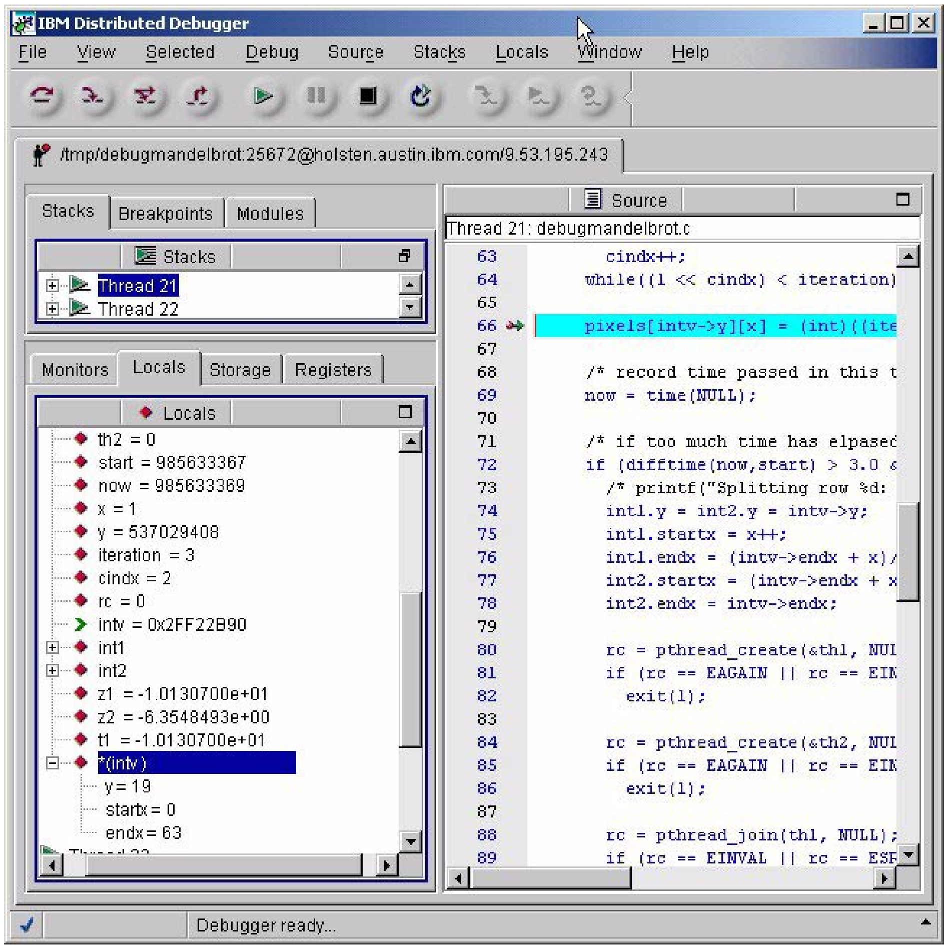

IBM Distributed Debugger
File View Selected Debug Source Stacks Locals Window Help
/tmp/debugmandelbrot:25672@holsten.austin.ibm.com/9.53.195.243
Stacks Breakpoints Modules
Stacks
Thread 21
Thread 22
Monitors Locals Storage Registers
Locals
Source
Thread 21: debugmandelbrot.c
Delete
Disable
Representation...
Edit
De-reference
Debugger ready...

```
Thread 21
    argv = 0x2FF22B90
    th1  = 0
    th2  = 0
    start = 985633367
    now  = 985633369
    x = 1
    y = 537029408
    iteration  = 3
    cindx = 2
    rc  = 0
    intv = 0x2FF22B90
    int1
    int2
    z1  = -1.0130700e+01
    z2  = -6.3548493e+00
    t1  = -1.0130700e+01
Thread 22
```

```
          cindx++;
        while((1 << cindx) < iteration)

        pixels[intv->y][x] = (int)((ite

        /* record time passed in this t
        now = time(NULL);

        /* if too much time has elpased
        if (difftime(now,start) > 3.0 &
          /* printf("Splitting row %d:
          int1.y = int2.y = intv->y;
          int1.startx = x++;
          int1.endx = (intv->endx + x)/
          int2.startx = (intv->endx + x
          int2.endx = intv->endx;

          rc = pthread_create(&th1, NUL
          if (rc == EAGAIN || rc == EIN
            exit(1);

          rc = pthread_create(&th2, NUL
          if (rc == EAGAIN || rc == EIN
            exit(1);

          rc = pthread_join(th1, NULL);
          if (rc == EINVAL || rc == ESR
```

Figure 77. Dereferencing a pointer value

We obtain the value of the interval (row number and endpoints) the specific thread has to compute, as shown in Figure 78 on page 448.

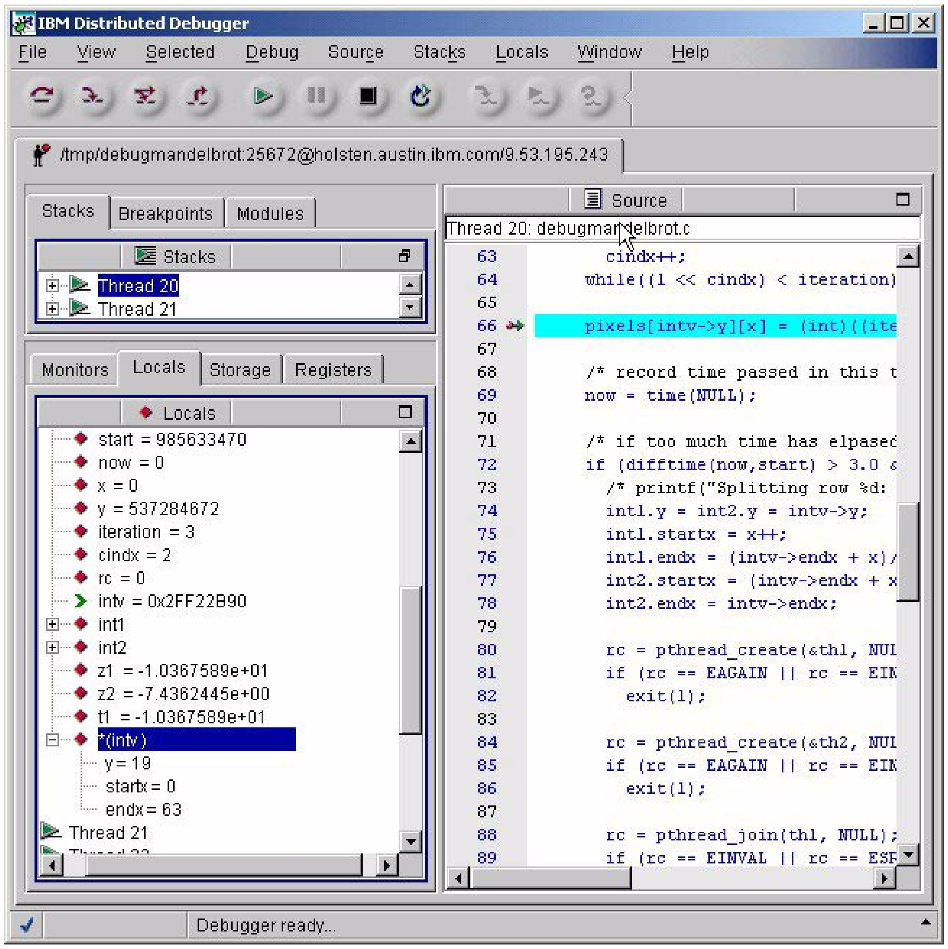


```
th2 = 0
start = 985633367
now = 985633369
x = 1
y = 537029408
iteration = 3
cindx = 2
rc = 0
intv = 0x2FF22B90
int1
int2
z1 = -1.0130700e+01
z2 = -6.3548493e+00
t1 = -1.0130700e+01
*(intv)
  y = 19
  startx = 0
  endx = 63
```

```
      cindx++;
    while((1 << cindx) < iteration)

    pixels[intv->y][x] = (int)((ite

    /* record time passed in this t
    now = time(NULL);

    /* if too much time has elpased
    if (difftime(now,start) > 3.0 &
      /* printf("Splitting row %d:
      int1.y = int2.y = intv->y;
      int1.startx = x++;
      int1.endx = (intv->endx + x)/
      int2.startx = (intv->endx + x
      int2.endx = intv->endx;

      rc = pthread_create(&th1, NUL
      if (rc == EAGAIN || rc == EIN
        exit(1);

      rc = pthread_create(&th2, NUL
      if (rc == EAGAIN || rc == EIN
        exit(1);

      rc = pthread_join(th1, NULL);
      if (rc == EINVAL || rc == ESR
```

Figure 78. Contents of a dereferenced structure pointer

We then press play again, until another thread gets scheduled. Again, we de-reference the intv variable for that thread, as shown in Figure 79 on page 449.


IBM Distributed Debugger
File View Selected Debug Source Stacks Locals Window Help
Stacks Breakpoints Modules
Monitors Locals Storage Registers
Source
Debugger ready...

/tmp/debugmandelbrot:25672@holsten.austin.ibm.com/9.53.195.243

Stacks
- Thread 20
- Thread 21

Locals
```
start = 985633470
now = 0
x = 0
y = 537284672
iteration = 3
cindx = 2
rc = 0
intv = 0x2FF22B90
int1
int2
z1 = -1.0367589e+01
z2 = -7.4362445e+00
t1 = -1.0367589e+01
*(intv)
  y = 19
  startx = 0
  endx = 63
Thread 21
```

Thread 20: debugmandelbrot.c
```
      cindx++;
    while((1 << cindx) < iteration)

    pixels[intv->y][x] = (int)((ite

    /* record time passed in this t
    now = time(NULL);

    /* if too much time has elpasec
    if (difftime(now,start) > 3.0 &
      /* printf("Splitting row %d:
      int1.y = int2.y = intv->y;
      int1.startx = x++;
      int1.endx = (intv->endx + x)/
      int2.startx = (intv->endx + x
      int2.endx = intv->endx;

      rc = pthread_create(&th1, NUL
      if (rc == EAGAIN || rc == EIN
        exit(1);

      rc = pthread_create(&th2, NUL
      if (rc == EAGAIN || rc == EIN
        exit(1);

      rc = pthread_join(th1, NULL);
      if (rc == EINVAL || rc == ESR
```

Figure 79. Structure contents for the next thread

We notice that the values are the same for different threads. We then realize the mistake. In line 127:

```
rc = pthread_create(&mythreads[y], NULL, row, (void *)&intv);
```

the threads are being created and asked to compute a unique row, based on the data pointed to by the reference &intv passed as argument. However, intv is being updated in line 123 - 125:

```
intv.y = y;
    intv.startx = 0;
    intv.endx = xres-1;
```

and since this variable is part of the shared process data, any thread can read the structure’s values, even as they are being updated. The solution is to pass these values to each thread, ensuring that they are not being overwritten. One way of solving the problem is shown in Appendix A.2.4, “mandelbrot4.c” on page 466.

The idebug system may seem a little complex when you use if for the first time. However the experienced developers that wrote this book can tell you that it is more than worth your while to take the time to learn how to use it. It is particularly good when debugging very large applications with source files distributed across many subdirectories.

Appendix A. Sample programs

This appendix contains listings of programs referred to in this book.

A.1 Makefile sample programs

This section contains code listings related to the `make` command.

A.1.1 The find_spec_targets_aix.ksh sample program

This is the find_spec_targets_aix.ksh sample program, that you can use to search makefiles for special targets that are *not* supported by the AIX 5L `make` command.

```
#!/usr/bin/ksh
########################################################
##
##  Sourcefile: find_spec_targets_aix.ksh
##
##  Description:
##              This script looks for special targets, that are not
##              supported by the make command in AIX.
##
##
##  Arguments  :
##              The name of a makefile to search in.
##
##  Change History:
##    Feature     Date        Who                          Description
##    xxxxxxx     02/25/2001  Jesper Frimann Ljungberg     Original version
##
##
##########################################################



ALL_SPEC_TARGETS=".DELETE_ON_ERROR .DONE .EXIT .EXPORT_ALL_VARIABLES
                .FAILED .GET_POSIX .INIT .INOBJECTDIR .INTERMEDIATE
                .INTERRUPT .KEEP_STATE .KEEP_STATE_FILE .MAIN
                .MAKE_VERSION .MUTEX .NOTPARALLEL .NO_PARALLEL .PARALLEL
                .PATH .PATHsuffix .PHONY .SCCS_GET .SCCS_GET_POSIX
                .SECONDARY .WAIT"

if [ "$*" = "" ] ; then
   OTHER_MAKEFILES=makefile
else
```

```
   OTHER_MAKEFILES=$*
fi

for SPECTARGET in $ALL_SPEC_TARGETS
do
    echo "Finding makefiles with $SPECTARGET special targets in them:"
    find . \( -name \*akefile\* -o -name \*AKEFILE\* -o -name
         $OTHER_MAKEFILES \) -exec egrep -l $SPECTARGET {} \;
done
```

A.1.2 The find_spec_targets_gnu.ksh sample program

This is the find_spec_targets_gnu.ksh sample program, that you can use to search makefiles for special targets that are *not* supported by the GNU `make` command.

```
#!/usr/bin/ksh
########################################################
##
##  Sourcefile: find_spec_targets_gnu.ksh
##
##  Description:
##              This script looks for special targets, that are not
##              supported by the gnu make command
##
##
##  Arguments  :
##              The name of a makefile to search in.
##
##  Change History:
##    Feature     Date        Who                          Description
##    xxxxxxx     02/25/2001  Jesper Frimann Ljungberg     Original version
##
##
##########################################################


ALL_SPEC_TARGETS=".DONE .EXIT .FAILED .GET_POSIX .INIT .INOBJECTDIR
                 .INTERRUPT .KEEP_STATE .KEEP_STATE_FILE .MAIN
                 .MAKE_VERSION .MUTEX .NO_PARALLEL .PARALLEL .PATH
                 .PATHsuffix.POSIX .SCCS_GET .SCCS_GET_POSIX .WAIT"


if [ "$*" = "" ] ; then
   OTHER_MAKEFILES=makefile
else
   OTHER_MAKEFILES=$*
```

```
fi

for SPECTARGET in $ALL_SPEC_TARGETS
do
    echo "Finding makefiles with $SPECTARGET special targets in it:"
    find . \( -name \*akefile\* -o -name \*AKEFILE\* -o -name\
    $OTHER_MAKEFILES \) -exec egrep -l $SPECTARGET  {} \;
done
```

A.1.3 The find_predef_macro_aix.ksh sample program

This is the find_predef_macro_aix.ksh sample program, that you can use to search makefiles for predefined macros that are *not* supported by the AIX 5L `make` command.

```
#!/usr/bin/ksh
########################################################
##
##  Sourcefile: find_predef_macro_aix.ksh
##
##  Description:
##              This script looks for Predefined macros,
##              that are not  supported by the make command in AIX.
##
##
##  Arguments  :
##              The name of a makefile to search in.
##
##  Change History:
##    Feature     Date        Who                         Description
##    xxxxxxx     02/25/2001  Jesper Frimann Ljungberg     Original version
##
##
#########################################################



ALL_SPEC_TARGETS='(CCC) (CP) (CPP) (CXX) (CWEAVE) (CTANGLE) (EC) (F77) (FC)
                 (LINT) (MACHINE) (MAKE_COMMAND)(MAKEFILE) (MAKEINFO) (MV)
                 (M2C) (PC) (RANLIB)(RC) (RM) (RMFLAGS) (TANGLE) (TEX)
                 (TEXI2DVI)(WEAVE) (YACCE) (YACCR)'


if [ "$*" = "" ] ; then
   OTHER_MAKEFILES=makefile
else
```

```
   OTHER_MAKEFILES=$*
fi

for SPECTARGET in $ALL_SPEC_TARGETS
do
    echo "Finding makefiles with $SPECTARGET special targets in it:"
    find . \( -name \*akefile\* -o -name \*AKEFILE\* -o -name
         $OTHER_MAKEFILES \) -exec grep -l $SPECTARGET {} \;
done
```

A.1.4 The find_predef_macro_gnu.ksh sample program

This is the find_predef_macro_gnu.ksh sample program, that you can use to search makefiles for predefined macros that are *not* supported by the GNU `make` command.

```
#!/usr/bin/ksh
########################################################
##
##  Sourcefile: find_predef_macro_gnu.ksh
##
##  Description:
##              This script looks for Predefined macros,
##              that are not supported by the gnu make command.
##
##
##  Arguments  :
##              The name of a makefile to search in.
##
##  Change History:
##    Feature     Date        Who                          Description
##    xxxxxxx     02/25/2001  Jesper Frimann Ljungberg     Original version
##
##
##########################################################



ALL_SPEC_TARGETS='(CCC) (CP)  (EC) (MACHINE) (MAKEFILE)(MV) (RANLIB) (RC)
                 (RMFLAGS) (WEAVE) (YACCR)'


if [ "$*" = "" ] ; then
   OTHER_MAKEFILES=makefile
else
   OTHER_MAKEFILES=$*
fi
```

```
for SPECTARGET in $ALL_SPEC_TARGETS
do
    echo "Finding makefiles with $SPECTARGET special targets in it:"
    find . \( -name \*akefile\* -o -name \*AKEFILE\* -o -name\
         $OTHER_MAKEFILES \) -exec grep -l $SPECTARGET {} \;
done
```

A.1.5 The find_internal_macro_aix.ksh sample program

This is the find_predef_macro_aix.ksh sample program, that you can use to search makefiles for internal macros that are *not* supported by the AIX 5L `make` command.

```
#!/usr/bin/ksh
########################################################
##
##  Sourcefile: find_internal_macro_aix.ksh
##
##  Description:
##              This script looks for internal macros,
##              that are not  supported by the make command in AIX.
##
##
##  Arguments   :
##              The name of a makefile to search in.
##
##  Change History:
##    Feature     Date        Who                           Description
##    xxxxxxx     02/26/2001  Jesper Frimann Ljungberg     Original version
##
##
#########################################################


if [ "$*" = "" ] ; then
   OTHER_MAKEFILES=makefile
else
   OTHER_MAKEFILES=$*
fi

echo 'Finding makefiles with the $+ internal macros in them:'
find . \( -name \*akefile\* -o -name \*AKEFILE\* -o -name
     $OTHER_MAKEFILES \) -exec grep -l '$+' {} \;
echo 'Finding  makefiles with the $^ internal macros in them:'
find . \( -name \*akefile\* -o -name \*AKEFILE\* -o -name
     $OTHER_MAKEFILES \) -exec grep -l '$^' {} \;
```

A.1.6 The hwinfo.c sample program

```
#include <stdio.h>
#include <sys/systemcfg.h>
/*
 * This program basically just some printfs to be used in makefiles,
 * or on the command line while compiling.
 * In a makefile it may be used like this:
 *
 * $cat makefile
 * XPORTFILES = $(B_EX) $(F_EX)
 * OBJFILES = foo.o bar.o
 * HWFLAGS=`hwinfo`
 * CFLAGS= $(HWFLAGS) -O3 -qstrict
 *
 * foo.o: ./src/foo.c ./inc/foo.h
 *        $(CC) $(CFLAGS) -c $(<D)/$*.c
 *
 * It have been found to work on a 43P,G40,S80,59H,H80,44P, and IA64b
 */

main()
{

  /*
   * Print out the -q<64/32>, after finding out if we are using a 32
   * or 64 bit processor
   *
   */

   printf(" -q%d",_system_configuration.width);

  /*
   *
   * Dertermine the processor and put some nice values to -qtune and -qarch.
   *
   */

   switch (_system_configuration.implementation) {
   case POWER_601:
      printf(" -qarch=601 -qtune=601");
      break;
   case POWER_603:
      printf(" -qarch=603 -qtune=603");
      break;
   case POWER_604:
      printf(" -qarch=604 -qtune=604");
      break;
   case POWER_620:
      printf("-qarch=rs64a -qtune=rs64a");
      break;
   case POWER_630:
      printf(" -qarch=pwr3 -qtune=pwr3");
      break;
   case POWER_A35:
      printf(" -qarch=403 -qtune=403");
      break;
   case POWER_RS64II:
      printf(" -qarch=rs64b -qtune=rs64b");
      break;
   case POWER_RS64III:
      printf(" -qarch=rs64c -qtune=rs64c");
      break;
   case POWER_RS64IV:
      printf(" -qarch=rs64c -qtune=rs64c");
      break;
   case POWER_RS2:
      printf(" -qarch=pwr2 -qtune=pwr2");
      break;
   case POWER_RS1:
      if (_system_configuration.rtc_type == RTC_IA64)
         printf(" -qarch=itanium -qtune=itanium");
      else
         printf(" -qarch=pwr -qtune=pwr");
      break;
   case POWER_RSC:
      if (_system_configuration.rtc_type == RTC_IA64)
         printf(" -qarch=itanium -qtune=itanium");
      else
         printf(" -qarch=p2sc -qtune=p2sc");
      break;
   }

  /*
   *   If we have a Level 1 instruction cache then print out the size,
   *   that it is X way associativ and line size.
   *
   */

   if (_system_configuration.icache_size > 0)
      printf(" -qcache=type=I:level=1:size=%d:assoc=%d:line=%d",
             _system_configuration.icache_size,
             _system_configuration.icache_asc,
             _system_configuration.icache_line);
  /*
   *   If we have a Level 1 data cache then print out the size, that it is
   *   X way associativ and line size.
   *
   */

   if (_system_configuration.dcache_size > 0)
      printf(" -qcache=type=D:level=1:size=%d:assoc=%d:line=%d",
             _system_configuration.dcache_size,
             _system_configuration.dcache_asc,
             _system_configuration.dcache_line);

  /*
   *   If we have a Level 2 cache then print out the size and
   *   that it is X way associativ.
   *
   */

   if (_system_configuration.L2_cache_size > 0)
      printf(" -qcache=type=C:level=2:size=%d:assoc=%d",
             _system_configuration.L2_cache_size,
             _system_configuration.L2_cache_asc);


  /*
   *   If we have a translation Lookaside buffer and a level 2 cache
   *   then print out the size of the instruction and data tlb and
   *   that it is X way associativ.
   */


   if (_system_configuration.L2_cache_size > 0 &&
       _system_configuration.tlb_attrib != 1 ) {
      printf(" -qcache=type=I:level=3:size=%d:assoc=%d",
             _system_configuration.itlb_size,
             _system_configuration.itlb_asc);
      printf(" -qcache=type=D:level=3:size=%d:assoc=%d",
             _system_configuration.dtlb_size,
             _system_configuration.dtlb_asc);
   }

  /*
   *   If we have a translation Lookaside buffer and NO! level
   *   2 cache then print out the size of the instruction and
   *   data tlb and that it is X  associativ.
```

```
   */

   if (_system_configuration.L2_cache_size == 0 &&
        _system_configuration.tlb_attrib != 1 ) {
      printf(" -qcache=type=I:level=2:size=%d:assoc=%d",
             _system_configuration.itlb_size,
             _system_configuration.itlb_asc);
      printf(" -qcache=type=D:level=2:size=%d:assoc=%d",
             _system_configuration.dtlb_size,
             _system_configuration.dtlb_asc);
   }


}
```

A.2 POSIX threads sample programs

This section contains the programs from Section 10.11, “Example: The Mandelbrot set” on page 391.

A.2.1 mandelbrot1.c

```
/*
 * File: mandelbrot1.c
 * Compile: cc -o mandelbrot1 mandelbrot1.c
 *  Description: Prints a ascii based picture
 *               of a part of the mandelbrot set.
 *
 * Arguments: none
 *
 * (C) COPYRIGHT International Business Machines Corp. 2001
 * All Rights Reserved
 *
 */

#include <stdlib.h>

int main(int argc, char **argv)
{
  printf("::::::::::::::::::::::::::::::::::::::::::::::::::::===");
  printf("====================o++==================::::::::::::::\n");
  printf(":::::::::::::::::::::::::::::::::::::::::::::::========");
  printf("====================++o++==+================:::::::::::\n");
  printf("::::::::::::::::::::::::::::::::::::::::::=============");
  printf("====================+++soo+++==================::::::::\n");
  printf(":::::::::::::::::::::::::::::::::::::==================");
  printf("==================+++Dooo++++=====================::::::\n");
  printf("::::::::::::::::::::::::::::::::::======================");
  printf("===============++Exoooo  oo+ooG=====================::::\n");
  printf("::::::::::::::::::::::::::::::==========================");
  printf("==============+++++o        oU++======================::\n");
  printf(":::::::::::::::::::::::::::=============================");
  printf("=======+++++++++++oO       o++N+++=====================:\n");
  printf(":::::::::::::::::::::===================================");
  printf("==++Dooo+++++Ox+OooooO   OooooO++oo+++++++o+============\n");
  printf(":::::::::::::::::::=====================================");
  printf("==E+++x  OoXox                    oo+++xooooR+==========\n");
  printf("::::::::::::::::========================================");
  printf("I+++++O                              o   @oA+===========\n");
  printf(":::::::::::::========================================R++");
  printf("xooooX                                  o+N++===========\n");
  printf(":::::::::::=====================++++======++======T+++++");
  printf("++x                                      o+G+o==========\n");
  printf("::::::::========================++o++++++++++++++++++++x");
  printf("X                                            OoE========\n");
  printf("::::::==========================+++++oxE+oox oooo++++++X");
  printf("                                          x+++==========\n");
  printf(":::==========================++++++Wo             oo++oo");
  printf("                                          : o+==========\n");
  printf("::========================++++++++oxX               Ooo ");
  printf("                                          O++===========\n");
  printf("========================+++++o+oooo                   x ");
  printf("                                          O+============\n");
  printf("============+++++++++++++++oo     @                   O ");
  printf("                                        +++=============\n");
  printf("============+++++++++++++++oo     @                   O ");
  printf("                                        +++=============\n");
  printf("========================+++++o+oooo                   x ");
  printf("                                          O+============\n");
  printf("::========================++++++++oxX               Ooo ");
  printf("                                          O++===========\n");
  printf(":::==========================+++++++o             oo++oo");
  printf("                                          : o+==========\n");
  printf("::::::==========================+++++oxx+oox oooo++++++X");
  printf("                                          x+++==========\n");
  printf("::::::::========================++o++++++++++++++++++++x");
  printf("X                                            Oo+========\n");
  printf(":::::::::::=====================+++========++======+++++");
  printf("++x                                      o+++o==========\n");
  printf(":::::::::::::=========================================++");
  printf("xooooX                                 o++++============\n");
  printf("::::::::::::::::========================================");
  printf("++++++O                              o   @o++===========\n");
```

```
  printf(":::::::::::::::::::=====================================");
  printf("==++++x  OoOox                    oo+++xoooo++==========\n");
  printf("::::::::::::::::::::::==================================");
  printf("==++xooo+++++Ox+OooooO   OooooO++oo+++++++o+============\n");
  printf("::::::::::::::::::::::::::==============================");
  printf("=======+++++++++++oO       o+++++======================:\n");
  printf("::::::::::::::::::::::::::::::==========================");
  printf("==============+++++o        o+++======================::\n");
  printf("::::::::::::::::::::::::::::::::::======================");
  printf("===============+++xoooo  oo+oo+=====================::::\n");
  printf("::::::::::::::::::::::::::::::::::::::==================");
  printf("==================++++ooo++++=====================::::::\n");
  printf(":::::::::::::::::::::::::::::::::::::::::::=============");
  printf("====================+++o+o+++===================::::::::\n");
  printf("::::::::::::::::::::::::::::::::::::::::::::::::========");
  printf("====================++o++==+=================:::::::::::\n");
  return 0;
}
```

A.2.2 mandelbrot2.c

```
/*
 * File: mandelbrot2.c
 * Compile: cc -o mandelbrot2 mandelbrot2.c
 * Description: Computes a ascii based picture
 *              of a specified part of the
 *              mandelbrot set.
 *
 * Arguments: none
 *
 * Change define values, then recompile:
 *     MAX_ITERATION, ..., RESOLUTION
 *
 * (C) COPYRIGHT International Business Machines Corp. 2001
 * All Rights Reserved
 *
 */

#include <stdlib.h>
#include <stdio.h>

/**************************************/
/* Values may be changed              */
/**************************************/
#define MAX_ITERATION 262144
#define MAX_LENGTH 100
#define X_MIN -2.1
```

```
#define Y_MIN -1.1
#define X_MAX 0.7
#define Y_MAX 1.1
#define RESOLUTION 35 /* vertical resolution, horizontal is then derived */

/****************************************/
/* Do not change the following          */
/****************************************/
#define COLORS 11
char *col = " -:=+oxOX@#";
int **pixels;
int x, y;
int xres = RESOLUTION*3.2;
int yres = RESOLUTION;
float xmin = X_MIN, ymin = Y_MIN;
float xstep = (X_MAX-X_MIN)/(RESOLUTION*3.2);
float ystep = (Y_MAX-Y_MIN)/RESOLUTION;

/****************************************/
/* compute row specified in y           */
/****************************************/
void row(int y)
{
  int x, iteration = 0, cindx = 0;
  float z1 = 0.0, z2 = 0.0, t1;

  for(x = 0; x < xres; x++, cindx = 0, iteration = 0, z1 = 0.0, z2 = 0.0) {
    /* compute one pixel */
    do {
      t1 = z1*z1-z2*z2+(xmin+x*xstep);
      z2 = 2*z1*z2+(ymin+y*ystep);
      z1 = t1;
      iteration++;
    }
    while(iteration < MAX_ITERATION && z1*z1+z2*z2 < MAX_LENGTH);

    do
      cindx++;
    while((1 << cindx) < iteration);

    pixels[y][x] = (int)((iteration >= MAX_ITERATION) ? 0 :
(cindx%COLORS));
  }
}

int main(int argc, char **argv)
{
  /* allocate memory for fractal pixels */
  if(NULL == (pixels = (int **)malloc(sizeof(int *)*(long)yres))) {
    perror("malloc");
    exit(1);
  }

  for(y = 0; y < yres; y++) {
    if(NULL == (pixels[y] = (int *)malloc(sizeof(int)*(long)xres))) {
      perror("malloc");
      exit(1);
    }
  }

  /* compute a row at the time */
  for(y = 0; y < yres; y++)
      row(y);

  /* print out fractal pixels */
  for(y = 0;y < yres;y++) {
    for(x = 0; x < xres; x++)
      putchar(col[pixels[y][x]]);
    putchar('\n');
  }

  /* free allocated memory */
  for(y = 0; y < yres; y++)
    free(pixels[y]);
  free(pixels);

  exit(0);
}
```

A.2.3 mandelbrot3.c

```
/*
 * File: mandelbrot3.c
 * Compile: cc_r -o mandelbrot3 mandelbrot3.c
 * Description: Computes a ascii based picture
 *              of a specified part of the
 *              mandelbrot set.
 *              Uses threads, one per vertical
 *              line.
 *
 * Arguments: none
 *
 * Change define values, then recompile:
 *     MAX_ITERATION, ..., RESOLUTION
 *
 * (C) COPYRIGHT International Business Machines Corp. 2001
 * All Rights Reserved
 *
 */

#include <pthread.h>
#include <stdlib.h>
#include <errno.h>

/**************************************/
/* Values may be changed              */
/**************************************/
#define MAX_ITERATION 262144
#define MAX_LENGTH 100
#define X_MIN -2.1
#define Y_MIN -1.1
#define X_MAX 0.7
#define Y_MAX 1.1
#define RESOLUTION 35 /* vertical resolution, horizontal is then derived */

/****************************************/
/* Do not change the following          */
/****************************************/
#define COLORS 11
char *col = " -:=+oxOX@#";
int **pixels;
int xres = RESOLUTION*3.2;
int yres = RESOLUTION;
float xmin = X_MIN, ymin = Y_MIN;
float xstep = (X_MAX-X_MIN)/(RESOLUTION*3.2);
float ystep = (Y_MAX-Y_MIN)/RESOLUTION;

/****************************************/
/* compute row specified in *argy       */
/****************************************/
void *row(void *argy)
{
  int x, iteration = 0, cindx = 0, y;
  float z1 = 0.0, z2 = 0.0, t1;

  y = *(int *)argy;

  for(x = 0; x < xres; x++, cindx = 0, iteration = 0, z1 = 0.0, z2 = 0.0) {
    /* compute one pixel */
    do {
      t1 = z1*z1-z2*z2+(xmin+x*xstep);
      z2 = 2*z1*z2+(ymin+y*ystep);
      z1 = t1;
      iteration++;
    }
    while(iteration < MAX_ITERATION && z1*z1+z2*z2 < MAX_LENGTH);

    do
      cindx++;
    while((1 << cindx) < iteration);

    pixels[y][x] = (iteration >= MAX_ITERATION) ? 0 : (cindx%COLORS);
  }
  pthread_exit(0);
}

int main(int argc, char **argv)
{
  pthread_t mythreads[RESOLUTION];
  int myarg[RESOLUTION];
  int x, y;
  int rc;

  /* allocate memory for fractal pixels */
  if(NULL == (pixels = (int **)malloc(sizeof(int *)*(long)yres))) {
    perror("malloc");
    exit(1);
  }

  for(y = 0; y < yres; y++)
    if(NULL == (pixels[y] = (int *)malloc(sizeof(int)*(long)xres))) {
      perror("malloc");
      exit(1);
    }

  /* start all row-threads */
  for(y = 0; y < yres; y++) {
    myarg[y] = y;
    rc = pthread_create(&mythreads[y], NULL, row, (void *)&myarg[y]);
    if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
      exit(1);
  }

  /* join all threads */
  for(y = 0; y < yres; y++) {
    rc = pthread_join(mythreads[y], NULL);
    if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
      exit(1);
  }

  /* print out fractal pixels */
  for(y = 0; y < yres; y++) {
    for(x = 0; x < xres; x++)
      putchar(col[pixels[y][x]]);
    putchar('\n');
  }

  /* free allocated memory */
  for(y = 0; y < yres; y++)
    free(pixels[y]);
  free(pixels);

  exit(0);
}
```

A.2.4 mandelbrot4.c

```
/*
 * File: mandelbrot4.c
 * Compile: cc_r -o mandelbrot4 mandelbrot4.c
 * Description: Computes a ascii based picture
 *              of a specified part of the
 *              mandelbrot set.
 *              Uses threads, one per vertical
 *              line. Each vertical line, starts
 *              new threads, if it records that
 *              it has been computing more than
 *              three seconds.
 *
 * Arguments: none
 *
 * Change define values, then recompile:
 *     MAX_ITERATION, ..., RESOLUTION
 *
 * (C) COPYRIGHT International Business Machines Corp. 2001
 * All Rights Reserved
 *
 */

#include <pthread.h>
#include <stdlib.h>
#include <errno.h>

/***************************************/
/* Values may be changed               */
```

```
/**************************************/
#define MAX_ITERATION 262144
#define MAX_LENGTH 100
#define X_MIN -2.1
#define Y_MIN -1.1
#define X_MAX 0.7
#define Y_MAX 1.1
#define RESOLUTION 35 /* vertical resolution, horizontal is then derived */

/****************************************/
/* Do not change the following variables */
/****************************************/
#define COLORS 11

typedef struct {
  int y, startx, endx;
} interval;

char *col = " -:=+oxOX@#";
int **pixels;
int xres = RESOLUTION*3.2;
int yres = RESOLUTION;
float xmin = X_MIN, ymin = Y_MIN;
float xstep = (X_MAX-X_MIN)/(RESOLUTION*3.2);
float ystep = (Y_MAX-Y_MIN)/RESOLUTION;

/***********************************************/
/* Compute row, subinterval specified in *argy   */
/* if computation takes more than 3 seconds      */
/* split into two threads                        */
/***********************************************/
void *row(void *argy)
{
  pthread_t th1, th2;
  time_t start, now;
  int x, y ,iteration = 0, cindx = 0;
  int rc;
  interval *intv, int1, int2;
  float z1 = 0.0, z2 = 0.0, t1;

  intv = (interval *)argy;

  /* record start time of this thread */
  start = time(NULL);

  for(x = intv->startx; x <= intv->endx; x++, cindx = 0, iteration = 0, z1
= 0.0, z2 = 0.0) {
    /* compute one pixel */
    do {
      t1 = z1*z1-z2*z2+(xmin+x*xstep);
      z2 = 2*z1*z2+(ymin+intv->y*ystep);
      z1 = t1;
      iteration++;
    }
    while(iteration < MAX_ITERATION && z1*z1+z2*z2 < MAX_LENGTH);

    do
      cindx++;
    while((1 << cindx) < iteration);

    pixels[intv->y][x] = (int)((iteration >= MAX_ITERATION) ? 0 :
(cindx%COLORS));

    /* record time passed in this thread */
    now = time(NULL);

    /* if too much time has elpased, start two new threads */
    if (difftime(now,start) > 3.0 && intv->endx-x > 3) {
     /* printf("Splitting row %d: (%d,%d)\n", intv->y, x+1, intv->endx); */
      int1.y = int2.y = intv->y;
      int1.startx = x++;
      int1.endx = (intv->endx + x)/2;
      int2.startx = (intv->endx + x)/2+1;
      int2.endx = intv->endx;

      rc = pthread_create(&th1, NULL, row, (void *)&int1);
      if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
   exit(1);

      rc = pthread_create(&th2, NULL, row, (void *)&int2);
      if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
   exit(1);

      rc = pthread_join(th1, NULL);
      if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
   exit(1);

      rc = pthread_join(th2, NULL);
      if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
   exit(1);

      x = intv->endx;
    }
  }
  pthread_exit(0);
}

int main(int argc, char **argv)
{
  pthread_t mythreads[RESOLUTION];
  interval intv[RESOLUTION];
  int x,y;
  int rc;

  /* allocate memory for fractal pixels */
  if(NULL == (pixels = (int **)malloc(sizeof(int *)*(long)yres))) {
    perror("malloc");
    exit(0);
  }

  for(y = 0; y < yres; y++)
    if(NULL == (pixels[y] = (int *)malloc(sizeof(int)*(long)xres))) {
      perror("malloc");
      exit(0);
    }

  /* start all row-threads */
  for(y = 0; y < yres; y++) {
    intv[y].y = y;
    intv[y].startx = 0;
    intv[y].endx = xres-1;

    rc = pthread_create(&mythreads[y], NULL, row, (void *)&intv[y]);
    if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
      exit(1);
  }

  /* join all threads */
  for(y = 0; y < yres; y++) {
    rc = pthread_join(mythreads[y], NULL);
    if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
      exit(1);
  }

  /* print out fractal pixels */
  for(y = 0;y < yres; y++) {
    for(x = 0; x < xres; x++)
      putchar(col[pixels[y][x]]);
    putchar('\n');
  }
  /* free allocated memory */
  for(y = 0; y < yres; y++)
    free(pixels[y]);
  free(pixels);

  exit(0);
}
```

A.2.5 mandelbrot5.c

```
/*
 * File: mandelbrot5.c
 * Compile: cc_r -o mandelbrot5 mandelbrot5.c
 * Description: Computes a ascii based picture
 *              of a specified part of the
 *              mandelbrot set.
 *              Uses threads, one per vertical
 *              line. Each vertical line, starts
 *              new threads, if it records that
 *              it has been computing more than
 *              three seconds and the processors
 *              are idle.
 *
 * Arguments: none
 *
 * Change define values, then recompile:
 *     MAX_ITERATION, ..., RESOLUTION
 *
 * (C) COPYRIGHT International Business Machines Corp. 2001
 * All Rights Reserved
 *
 */

#include <pthread.h>
#include <stdlib.h>
#include <sys/sysinfo.h>
#include <nlist.h>
#include <sys/systemcfg.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>

/***************************************/
/* Values may be changed               */
/***************************************/
#define MAX_ITERATION 262144
```

```
#define MAX_LENGTH 100
#define X_MIN -2.1
#define Y_MIN -1.1
#define X_MAX 0.7
#define Y_MAX 1.1
#define RESOLUTION 35 /* vertical resolution, horizontal is then derived */

/*****************************************/
/* Do not change the following           */
/*****************************************/
#define N_VALUE(index) (nlists[index].n_value)
#define NLIST_CPU       5
#define NUMBER_OF_KNSTRUCTS 6
#define COLORS 11

typedef struct {
  int y, startx, endx;
} interval;

int knlist(struct nlist *NList, int NumberOfElements, int Size);
int cpus, fd;

struct nlist nlists[7] = {
  { "sysinfo",    (long)0, (short)0, (unsigned short)0, (char)0, (char)0 },
  { "vmker",      (long)0, (short)0, (unsigned short)0, (char)0, (char)0 },
  { "vmminfo",    (long)0, (short)0, (unsigned short)0, (char)0, (char)0 },
  { "iostat",     (long)0, (short)0, (unsigned short)0, (char)0, (char)0 },
  { "ifnet",      (long)0, (short)0, (unsigned short)0, (char)0, (char)0 },
  { "cpuinfo",    (long)0, (short)0, (unsigned short)0, (char)0, (char)0 },
  { NULL,         (long)0, (short)0, (unsigned short)0, (char)0, (char)0 }
};

char *col = " -:=+oxOX@#";
int **pixels;
int xres = RESOLUTION*3.2;
int yres = RESOLUTION;
float xmin = X_MIN, ymin = Y_MIN;
float xstep = (X_MAX-X_MIN)/(RESOLUTION*3.2);
float ystep = (Y_MAX-Y_MIN)/RESOLUTION;

/**********************/
/* read kernel memory */
/**********************/
void read_kmem(char *buf, long bufsize, long n_value)
{
  /* Get the structure from the running kernel.  */
  if (lseek(fd, n_value, SEEK_SET) == -1) {
    perror("lseek error");
    exit(1);
  }
  if (read(fd, buf, bufsize) == -1) {
    perror("read error");
    exit(1);
  }
}

/************************************************/
/* Compute row, subinterval specified in *argy   */
/* if computation takes more than 3 seconds, and */
/* some cpus seem idle, split into two threads   */
/************************************************/
void *row(void *argy)
{
  struct cpuinfo *cpuinfoa, *cpuinfob;
  long cpu_idle, cpu_user, cpu_sys, cpu_wait, cpu_sum, i;

  pthread_t th1, th2;
  time_t start, now;
  int x, iteration = 0, cindx = 0;
  int rc;
  interval *intv, int1, int2;
  float z1 = 0.0, z2 = 0.0, t1;
  double idlemax;

  /* used for keeping track of CPU usage */
  cpuinfoa = (struct cpuinfo *)malloc(sizeof(struct cpuinfo) * (long)cpus);
  cpuinfob = (struct cpuinfo *)malloc(sizeof(struct cpuinfo) * (long)cpus);

  intv = (interval *)argy;

  /* record start time of this thread */
  start = time(NULL);

  for(x = intv->startx; x <= intv->endx; x++, cindx = 0, iteration = 0, z1
= 0.0, z2 = 0.0) {
    read_kmem((char *)&cpuinfoa[0], sizeof(struct cpuinfo)*(long)cpus,
N_VALUE(NLIST_CPU));

    /* compute one pixel */
    do {
      t1 = z1*z1-z2*z2+(xmin+x*xstep);
      z2 = 2*z1*z2+(ymin+intv->y*ystep);
      z1 = t1;
      iteration++;
    }
    while(iteration < MAX_ITERATION && z1*z1+z2*z2 < MAX_LENGTH);

    do
      cindx++;
    while((1 << cindx) < iteration);

    pixels[intv->y][x] = (iteration >= MAX_ITERATION) ? 0 : (cindx%COLORS);

    /* sample cpu usage */
    read_kmem((char *)&cpuinfob[0], sizeof(struct cpuinfo )*(long)cpus,
N_VALUE(NLIST_CPU));

    idlemax = 0.0;

    for (i=0; i<cpus; i++) {
      cpu_idle = cpuinfob[i].cpu[CPU_IDLE]-cpuinfoa[i].cpu[CPU_IDLE];
      cpu_user = cpuinfob[i].cpu[CPU_USER]-cpuinfoa[i].cpu[CPU_USER];
      cpu_sys  = cpuinfob[i].cpu[CPU_KERNEL]-cpuinfoa[i].cpu[CPU_KERNEL];
      cpu_wait = cpuinfob[i].cpu[CPU_WAIT]-cpuinfoa[i].cpu[CPU_WAIT];
      cpu_sum = cpu_idle + cpu_user + cpu_sys + cpu_wait;
      if ((double)cpu_idle/(double)cpu_sum*100.0 > idlemax)
   idlemax = (double)cpu_idle/(double)cpu_sum*100.0;
    }

      /* record time passed in this thread */
    now = time(NULL);

    /* if too much time has elpased and user cpu is too idle, start two new
threads */
    if (difftime(now,start) > 3.0 && intv->endx-x > 3 && idlemax > 20.0) {
   /* printf("Splitting row %d: (%d,%d)\n", intv->y, x+1, intv->endx); */
      int1.y = int2.y = intv->y;
      int1.startx = x++;
      int1.endx = (intv->endx + x)/2;
      int2.startx = (intv->endx + x)/2+1;
      int2.endx = intv->endx;

      rc = pthread_create(&th1, NULL, row, (void *)&int1);
      if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
   exit(1);

      rc = pthread_create(&th2, NULL, row, (void *)&int2);
      if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
   exit(1);

      rc = pthread_join(th1, NULL);
```

```
       if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
    exit(1);

       rc = pthread_join(th2, NULL);
       if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
    exit(1);

       x = intv->endx;
     }
   }
   pthread_exit(0);
 }

 int main(int argc, char **argv)
 {
   pthread_t mythreads[RESOLUTION];
   interval intv[RESOLUTION];
   struct cpuinfo *cpuinfoa, *cpuinfob;
   long cpu_idle, cpu_user, cpu_sys, cpu_wait, cpu_sum;
   int x, y;
   int rc;

   /* Get the number of CPUs */
   cpus = _system_configuration.ncpus;

   /* Request the Kernel addresses - if this fails stop now */
   if (knlist(nlists, NUMBER_OF_KNSTRUCTS, sizeof(struct nlist )) == -1)
     exit(1);

   /* Open kernel memory for reading */
   if ((fd = open("/dev/kmem", O_RDONLY)) == -1) {
     perror("opening /dev/kmem");
     printf("As root use: chmod ugo+r /dev/kmem\n");
     printf("to fix this\n");
     exit(1);
   }

   /* allocate memory for fractal pixels */
   if(NULL == (pixels = (int **)malloc(sizeof(int *)*(long)yres))) {
     perror("malloc");
     exit(0);
   }

   for(y = 0; y < yres; y++)
     if(NULL == (pixels[y] = (int *)malloc(sizeof(int)*(long)xres))) {
       perror("malloc");
       exit(0);
     }

   /* start all row-threads */
   for(y = 0; y < yres; y++) {
     intv[y].y = y;
     intv[y].startx = 0;
     intv[y].endx = xres-1;

     rc = pthread_create(&mythreads[y], NULL, row, (void *)&intv[y]);
     if (rc == EAGAIN || rc == EINVAL || rc == EPERM)
       exit(1);
   }

   /* join all threads */
   for(y = 0; y < yres; y++) {
     rc = pthread_join(mythreads[y], NULL);
     if (rc == EINVAL || rc == ESRCH || rc == EDEADLK)
       exit(1);
   }

   /* print out fractal pixels */
   for(y = 0;y < yres; y++) {
     for(x = 0; x < xres; x++)
       putchar(col[pixels[y][x]]);
     putchar('\n');
   }

   /* free allocated memory */
   for(y = 0; y < yres; y++)
     free(pixels[y]);
   free(pixels);

   exit(0);
 }
```

Appendix B. Default inference rules for the make commands

This appendix contains information on the default inference rules for the different `make` commands available on various UNIX-based platforms, along with those defined by the POSIX standard (IEEE Std 1003.2) and the GNU `make` command. When you see a - (hyphen) in the tables, this means that the particular rule is not implemented on the platform in question.

The rules have been split into tables for ease of use, and have been sorted in alphabetic order, ordered by the source. Thus, if you want to find the rule that governs how to make a .o file from a .c file, you look after the .c.o rule, where the .c is what the tables are sorted by.

Intermediate variables have been expanded. So if you are using the value of a variable like COMPILE.c in your makefiles (to redefine the way the compiler is called or which compiler is to be used), then you have to do some substitution in the rules. For example, on Solaris, the double suffix rule for making a .o file from a .c file looks like this:

```
$(CC) $(CFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $<
```

This is the version that is listed in Table 111 on page 491. This rule is a result of a merger of the rules:

```
$(COMPILE.c) $(OUTPUT_OPTION) $<
COMPILE.c= $(CC) $(CFLAGS) $(CPPFLAGS) -c
```

These mergers have been made so that you can compare the different rules without having to cross-reference other tables to look up intermediate variables.

B.1 Single suffix inference rules

Table 110 lists the single suffix inference rules.

Table 110. Single suffix rules

Suffix	Make	Single suffix rule
.a~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`

Suffix	Make	Single suffix rule
	POSIX	-
.c	AIX	`$(CC) $(CFLAGS) $(LDFLAGS) $< -o $@`
	HP-UX	`$(CC) $(CFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	`$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)-o $@ $< $(LDLIBD)`
	Tru64	`$(CC) $(LDFLAGS) $(CFLAGS) $< $(LOADLIBES) -o $@`
	GNU	`$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) $^ $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	`$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<`

.c~	AIX	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) $(CFLAGS) $(LDFLAGS) $*.c -o $*`
		`-rm -f $*.c`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) $(CFLAGS) $(LDFLAGS) -o $* $*.c`
		`-rm -f $*.c`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.C	AIX	`$(CCC) $(CCFLAGS) $(LDFLAGS) $< -o $@`
	HP-UX	`$(CXX) $(CXXFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	`$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	-
	GNU	`$(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) $^ $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	-

.C~	AIX	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CCC) $(CCFLAGS) $(LDFLAGS) $*.C -o $*`
		`-rm -f $*.C`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CXX) $(CXXFLAGS) $(LDFLAGS) $*.C -o $*`
		`rm -f $*.C`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS) -o $@ $*.C $(LDLIBS)`

	Tru64	-

Suffix	Make	Single suffix rule
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.cc	AIX	-
	HP-UX	`$(CXX) $(CXXFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	`$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	-
	GNU	`$(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	-
.cc~	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cc`
		`$(CXX) $(CXXFLAGS) $(LDFLAGS) $*.cc -o $*`
		`rm -f $*.cc`

	Solaris	`$(GET) $(GFLAGS) -p $, > $*.cc`
		`$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)-o $@ $*.cc $(LDLIBS)`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.ch~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.CLEAN UP	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(RM) $(RMFLAGS) $?`
	GNU	-

Suffix	Make	Single suffix rule
	POSIX	-
.CO	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(CO) $(COFLAGS) $< $@`
	GNU	-
	POSIX	-
.cpp	AIX	-
	HP-UX	`$(CXX) $(CXXFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	-
	Tru64	-

	GNU	`$(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) $^`
		`$(LOADLIBES) $(LDLIBS) -o $@`

	POSIX	-
.cpp~	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cpp`
		`$(CXX) $(CXXFLAGS) $(LDFLAGS) $*.cpp -o $*`
		`rm -f $*.cpp`

	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.csh	AIX	-
	HP-UX	-
	Solaris	-

	Tru64	`$(CP) $(CPFLAGS) $< $@`
		`chmod +x $@`

	GNU	-
	POSIX	-
.cxx	AIX	-

Suffix	Make	Single suffix rule
	HP-UX	`$(CXX) $(CXXFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cxx~	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cxx`
		`$(CXX) $(CXXFLAGS) $(LDFLAGS) $*.cxx -o $*`
		`rm -f $*.cxx`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.def~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.dvi~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.e	AIX	-
	HP-UX	-
	Solaris	-

Suffix	Make	Single suffix rule
	Tru64	`$(EC) $(LDFLAGS) $(EFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-
.el~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.elc	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.f	AIX	`$(FC) $(FFLAGS) $(LDFLAGS) $< -o $@`
	HP-UX	`$(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<`
	Solaris	`$(FC) $(FFLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	`$(FC) $(LDFLAGS) $(FFLAGS) $< $(LOADLIBES) -o $@`
	GNU	`$(FC) $(FFLAGS) $(LDFLAGS) $(TARGET_ARCH) $^ $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	`$(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<`

.f~	AIX	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) $(LDFLAGS) $*.f -o $*`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) $(LDFLAGS) -o $* $*.f`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`

Suffix	Make	Single suffix rule
	POSIX	-
.f90	AIX	-
	HP-UX	-
	Solaris	`$(F90C) $(F90FLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	`$(F90C) $(FFLAGS) $(LDFLAGS) -o $@`
	GNU	-
	POSIX	-
.f90~	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.f90`
		`$(F90C) $(F90FLAGS) $(LDFLAGS)-o $@ $*.f90 $(LDLIBS)`

	Tru64	`$(GET) $(GFLAGS) -p $< > $*.f90`
		`$(F90C) $(FFLAGS) $(LDFLAGS) -o $@ $*.f90`

	GNU	-
	POSIX	-
.ftn	AIX	-
	HP-UX	-
	Solaris	`$(F90C) $(F90FLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	-
	GNU	-
	POSIX	-
.ftn~	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.ftn`
		`$(F90C) $(F90FLAGS) $(LDFLAGS) -o $@ $*.ftn $(LDLIBS)`

	Tru64	-
	GNU	-
	POSIX	-
.F	AIX	-

Suffix	Make	Single suffix rule
	HP-UX	`$(FC) $(FFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	`$(FC) $(FFLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	`$(FC) $(LDFLAGS) $(FFLAGS) $< $(LOADLIBES) -o $@`

	GNU	`$(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) $^`
		`$(LOADLIBES) $(LDLIBS) -o $@`

	POSIX	-
.F~	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.F`
		`$(FC) $(FFLAGS) $(LDFLAGS) $*.F -o $*`
		`-rm -f $*.F`

	Solaris	`$(GET) $(GFLAGS) -p $< >.$*.F`
		`$(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.F`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.h~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.info~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.l	AIX	-
	HP-UX	-

Suffix	Make	Single suffix rule

	Solaris	`$(RM) $*.c`
		`$(LEX) $(LFLAGS) -t $< > $*.c`
		`$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) -o $@ $*.c -ll $(LDLIBS)`

	Tru64	`$(LEX) $(LFLAGS) $<`
		`$(CC) $(LDFLAGS) $(CFLAGS) lex.yy.c $(LOADLIBES) -ll -o $@`
		`$(RM) $(RMFLAGS) lex.yy.c`

	GNU	-
	POSIX	-
.l~	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.l`
		`$(LEX) $(LFLAGS) $*.l`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`rm -f lex.yy.c`
		`mv lex.yy.c $@`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.ln~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
makefile	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.mod	AIX	-

Suffix	Make	Single suffix rule
	HP-UX	-
	Solaris	`$(M2C) $(M2FLAGS) $(MODFLAGS)-o $@ -e $@ $<`
	Tru64	-
	GNU	`$(M2C) $(M2FLAGS) $(MODFLAGS) $(TARGET_ARCH) -o $@ -e $@ $^`
	POSIX	-
.mod~	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.mod`
		`$(M2C) $(M2FLAGS) $(MODFLAGS) -o $@ -e $@ $*.mod`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.o~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.out~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.p	AIX	-
	HP-UX	`$(PC) $(PFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	`$(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	`$(PC) $(LDFLAGS) $(PFLAGS) $< $(LOADLIBES) -o $@`

Suffix	Make	Single suffix rule
	GNU	`$(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) $^ $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	-
.p~	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.p`
		`$(PC) $(PFLAGS) $(LDFLAGS) $*.p -o $*`
		`-rm -f $*.p`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.p`
		`$(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS) -o $@ $*.p $(LDLIBS)`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.r	AIX	-
	HP-UX	`$(FC) $(RFLAGS) $< $(LDFLAGS) -o $@`
	Solaris	`$(FC) $(FFLAGS) $(RFLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
	Tru64	`$(RC) $(LDFLAGS) $(RFLAGS) $< $(LOADLIBES) -o $@`
	GNU	`$(FC) $(FFLAGS) $(RFLAGS) $(LDFLAGS) $(TARGET_ARCH) $^ $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	-
.r~	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.r`
		`$(FC) $(RFLAGS) $(LDFLAGS) $*.r -o $*`
		`-rm -f $*.r`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.r`
		`$(FC) $(FFLAGS) $(RFLAGS) $(LDFLAGS) -o $@ $*.r $(LDLIBS)`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.s	AIX	-
	HP-UX	-
	Solaris	`-s`
	Tru64	`$(AS) $(ASFLAGS) -o $@ $<`

Suffix	Make	Single suffix rule
	GNU	`$(CC) $(LDFLAGS) $(TARGET_MACH) $^ $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	-
.s~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.S	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(CC) $(LDFLAGS) $(TARGET_MACH) $^ $(LOADLIBES) $(LDLIBS) -o $@`
	POSIX	-
.S~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	GNU	-
	POSIX	-
.SCCS_ GET	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@`
	GNU	-
	POSIX	-

Suffix	Make	Single suffix rule
.sh	AIX	`cp $< $@; chmod a+x $@`
	HP-UX	`cp $< $@; chmod 0777 $@`

	Solaris	`$(RM) $@`
		`cat $< > $@`
		`chmod -x $@`

	Tru64	`$(CP) $(CPFLAGS) $< $@`
		`chmod +x $@`

	GNU	`cat $< > $@`
		`chmod a+x $@`

	POSIX	`cp $< $@`
		`chmod a+x $@`

.sh~	AIX	`$(GET) $(GFLAGS) -p $< > $*.sh`
		`cp $*.sh $*; chmod a+x $@`
		`-rm -f $*.c`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.sh`
		`cp $*.sh $*; chmod 0777 $@`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.sh`
		`cp $* $*.sh $@`
		`chmod a+x $@`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	
.sym~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.tex~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`

Suffix	Make	Single suffix rule
	POSIX	-
.texi~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.texinfo ~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.txinfo	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.w~	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.web~	AIX	-

Suffix	Make	Single suffix rule
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-
.y	AIX	-
	HP-UX	-

	Solaris	`$(YACC) $(YFLAGS) $<`
		`$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) -o $@ $< $(LDLIBS)`
		`$(RM) ytab.c`

	Tru64	`$(YACC) $(YFLAGS) $<`
		`$(CC) $(LDFLAGS) $(CFLAGS) y.tab.c $(LOADLIBES) -ly -o $@`
		`$(RM) $(RMFLAGS) y.tab.c`

	GNU	-
	POSIX	-
.y~	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.y`
		`$(YACC) $(YFLAGS) $*.y`
		`$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $@ ytab.c`
		`$(RM) y.tab.c`

	Tru64	-
	GNU	`$(GET) $(GFLAGS) $(SCCS_OUTPUT_OPTION) $<`
	POSIX	-

B.2 Double suffix inference rules

Table 111 lists the double suffix inference rules.

Table 111. Double suffix rules

Suffix	Make	Double suffix rule

.c.a	AIX	`$(CC) -c $(CFLAGS) $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.o`

Suffix	Make	Double suffix rule

	HP-UX	`$(CC) -c $(CFLAGS) $<`
		`ar rv $@ $*.o`
		`rm -f $*.o`

	Solaris	`$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	`$(CC) $(CFLAGS) -c $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`$(RM) $(RMFLAGS) $*.o`

	GNU	-

	POSIX	`$(CC) -c $(CFLAGS) $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.o`

.c~.a	AIX	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) -c $(CFLAGS) $*.c`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.[co]`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) -c $(CFLAGS) $*.c`
		`ar rv $@ $*.o`
		`rm -f $*.[co]`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $% $*.c`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.c~.c	AIX	`$(GET) $(GFLAGS) -p $< > $*.c`
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.c`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.c.o	AIX	`$(CC) $(CFLAGS) -c $<`
	HP-UX	`$(CC) $(CFLAGS) -c $<`
	Solaris	`$(CC) $(CFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $<`
	Tru64	`$(CC) $(CFLAGS) -c $<`

Suffix	Make	Double suffix rule
	GNU	`$(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c $(OUTPUT_OPTION) $<`
	POSIX	`$(CC) $(CFLAGS) -c $<`
.c.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(CC) $(LDFLAGS) $(CFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-

.c~.o	AIX	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) $(CFLAGS) -c $*.c`
		`-rm -f $*.c`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) $(CFLAGS) -c $*.c`
		`-rm -f $*.c`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(CC) $(CFLAGS) -c  $*.c`

	Tru64	-
	GNU	-
	POSIX	-
.c.ln	AIX	-
	HP-UX	-
	Solaris	`$(LINT) $(LINTFLAGS) $(CPPFLAGS) $(OUTPUT_OPTION) -c $<`
	Tru64	-
	GNU	`$(LINT) $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -C$* $<`
	POSIX	-
.c~.ln	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.c`
		`$(LINT) $(LINTFLAGS) $(CPPFLAGS) $(OUTPUT_OPTION) -c $*.c`

	Tru64	-

Suffix	Make	Double suffix rule
	GNU	-
	POSIX	-
.cc.a	AIX	-

	HP-UX	`$(CXX) -c $(CXXFLAGS) $<`
		`ar rv $@ $*.o`
		`rm -f $*.o`

	Solaris	`$(CCC) $(CCFLAGS) $(CPPFLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.cc~.a	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cc`
		`$(CXX) -c $(CXXFLAGS) $*.cc`
		`ar rv $@ $*.o`
		`rm -f $*.cc $*.o`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.cc`
		`$(CCC) $(CCFLAGS) $(CPPFLAGS) -c -o $% $*.cc`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.cc.o	AIX	-
	HP-UX	`$(CXX) $(CXXFLAGS) -c $<`
	Solaris	`$(CCC) $(CCFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $<`
	Tru64	-

	GNU	`$(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c`
		`$(OUTPUT_OPTION) $<`

	POSIX	-
.cc.cc~	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cc`
	Solaris	-

Suffix	Make	Double suffix rule
	Tru64	-
	GNU	-
	POSIX	-
.cc~.o	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cc`
		`$(CXX) $(CXXFLAGS) -c $*.cc`
		`rm -f $*.cc`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.cc`
		`$(CCC) $(CCFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $*.cc`

	Tru64	-
	GNU	-
	POSIX	-
.cpp.a	AIX	-

	HP-UX	`$(CXX) -c $(CXXFLAGS) $<`
		`ar rv $@ $*.o`
		`rm -f $*.o`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cpp~.a	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cpp`
		`$(CXX) -c $(CXXFLAGS) $*.cpp`
		`ar rv $@ $*.o`
		`rm -f $*.cpp $*.o`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cpp~. cpp	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cpp`

Suffix	Make	Double suffix rule
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cpp.o	AIX	-
	HP-UX	`$(CXX) $(CXXFLAGS) -c $<`
	Solaris	-
	Tru64	-
	GNU	`$(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c $(OUTPUT_OPTION) $<`
	POSIX	-
.cpp~.o	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cpp`
		`$(CXX) $(CXXFLAGS) -c $*.cpp`
		`rm -f $*.cpp`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cps.h	AIX	-
	HP-UX	-
	Solaris	`$(CPS) $(CPSFLAGS) $*.cps`
	Tru64	-
	GNU	-
	POSIX	-
.cps~.h	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.cps`
		`$(CPS) $(CPSFLAGS) $*.cps`

	Tru64	-

Suffix	Make	Double suffix rule
	GNU	-
	POSIX	-
.cxx.a	AIX	-

	HP-UX	`$(CXX) -c $(CXXFLAGS) $<`
		`ar rv $@ $*.o`
		`rm -f $*.o`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cxx~.a	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cxx`
		`$(CXX) -c $(CXXFLAGS) $*.cxx`
		`ar rv $@ $*.o`
		`rm -f $*.cxx $*.o`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cxx~. cxx	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cxx`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.cxx.o	AIX	-
	HP-UX	`$(CXX) $(CXXFLAGS) -c $<`
	Solaris	-
	Tru64	-

Suffix	Make	Double suffix rule
	GNU	-
	POSIX	-
.cxx~.o	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.cxx`
		`$(CXX) $(CXXFLAGS) -c $*.cxx`
		`rm -f $*.cxx`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-

.C.a	AIX	`$(CCC) -c $(CCFLAGS) $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.o`

	HP-UX	`$(CXX) -c $(CXXFLAGS) $<`
		`ar rv $@ $*.o`
		`rm -f $*.o`

	Solaris	`$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-

.C~.a	AIX	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CCC) -c $(CCFLAGS) $*.C`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.[Co]`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CXX) -c $(CXXFLAGS) $*.C`
		`ar rv $@ $*.o`
		`rm -f $*.[Co]`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $% $*.C`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-

Suffix	Make	Double suffix rule
.C~.C	AIX	`$(GET) $(GFLAGS) -p $< > $*.C; chmod 444 $*.C`
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.C`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.C.o	AIX	`$(CCC) $(CCFLAGS) -c $`
	HP-UX	`$(CXX) $(CXXFLAGS) -c $<`
	Solaris	`$(CC) $(CFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $<`
	Tru64	-
	GNU	`$(CXX) $(CXXFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c $(OUTPUT_OPTION) $<`
	POSIX	-

.C~.o	AIX	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CCC) $(CCFLAGS) -c $`
		`-rm -f $*.C`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CXX) $(CXXFLAGS) -c $*.C`
		`rm -f $*.C`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.C`
		`$(CC) $(CFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $*.C`

	Tru64	-
	GNU	-
	POSIX	-
.def. sym	AIX	-
	HP-UX	-
	Solaris	`$(M2C) $(M2FLAGS) $(DEFFLAGS) -o $@ $<`
	Tru64	-
	GNU	`$(M2C) $(M2FLAGS) $(DEFFLAGS) $(TARGET_ARCH) -o $@ $<`
	POSIX	-

Suffix	Make	Double suffix rule
.def~.sym	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.def`
		`$(M2C) $(M2FLAGS) $(DEFFLAGS) -o $@ $*.def`

	Tru64	-
	GNU	-
	POSIX	-
.e.o	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(EC) $(EFLAGS) -c $<`
	GNU	-
	POSIX	-
.e.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(EC) $(LDFLAGS) $(EFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-

.f.a	AIX	`$(FC) -c $(FFLAGS) $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.o`

	HP-UX	`$(FC) -c $(FFLAGS) $<`
		`ar rv $@ $*.o`
		`rm -f $*.o`

	Solaris	`$(FC) $(FFLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	`$(FC) $(FFLAGS) -c $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`$(RM) $(RMFLAGS) $*.o`

	GNU	-

Suffix	Make	Double suffix rule

	POSIX	`$(FC) $(FFLAGS) $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.o`

.f~.a	AIX	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) -c $(FFLAGS) $*.f`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`rm -f $*.[fo]`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) -c $(FFLAGS) $*.f`
		`ar rv $@ $*.o`
		`rm -f $*.[fo]`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) -c -o $% $*.f`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.f~.f	AIX	`$(GET) $(GFLAGS) -p $< > $@`
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.f`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.f.o	AIX	`$(FC) $(FFLAGS) -c $<`
	HP-UX	`$(FC) $(FFLAGS) -c $<`
	Solaris	`$(FC) $(FFLAGS) -c $(OUTPUT_OPTION) $<`
	Tru64	`$(FC) $(FFLAGS) -c $<`
	GNU	`$(FC) $(FFLAGS) $(TARGET_ARCH) -c $(OUTPUT_OPTION) $<`
	POSIX	`$(FC) $(FFLAGS) -c $<`
.f.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(FC) $(LDFLAGS) $(FFLAGS) $< $(LOADLIBES) -o $@`

Suffix	Make	Double suffix rule
	GNU	-
	POSIX	-

.f~.o	AIX	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) -c $*.f`
		`rm -f $*.f`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) -c $*.f`
		`-rm -f $*.f`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) -c  $*.f`

	Tru64	`$(GET) $(GFLAGS) -p $< > $*.f`
		`$(FC) $(FFLAGS) -c $*.f`

	GNU	-
	POSIX	-
.f90.a	AIX	-
	HP-UX	-

	Solaris	`$(F90C) $(F90FLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	`$(F90C) $(FFLAGS) -c $<`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`$(RM) $(RMFLAGS) $*.o`

	GNU	-
	POSIX	-
.f90~.a	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.f90`
		`$(F90C) $(F90FLAGS) -c -o $% $*.f90`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.f90~.mod	AIX	-

Suffix	Make	Double suffix rule
	HP-UX	-
	Solaris	-

	Tru64	`$(GET) $(GFLAGS) -p $< > $*.f90`
		`$(F90C) $(FFLAGS) -c $*.f90`

	GNU	-
	POSIX	-
.f90.o	AIX	-
	HP-UX	-
	Solaris	`$(F90C) $(F90FLAGS) -c $(OUTPUT_OPTION) $<`
	Tru64	-
	GNU	-
	POSIX	-
.f90~.o	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.f90`
		`$(F90C) $(F90FLAGS) -c $(OUTPUT_OPTION) $*.f90`

	Tru64	`$(GET) $(GFLAGS) -p $< > $*.f90`
		`$(F90C) $(FFLAGS) -c $*.f90`

	GNU	-
	POSIX	-
.F.a	AIX	-
	HP-UX	-

	Solaris	`$(FC) $(FFLAGS) $(CPPFLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.F~.a	AIX	-
	HP-UX	-

Suffix	Make	Double suffix rule

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.F`
		`$(FC) $(FFLAGS) $(CPPFLAGS) -c -o $% $*.F`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.F.f	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-

	GNU	`$(FC) $(FFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -F`
		`$(OUTPUT_OPTION) $<`

	POSIX	-
.F.o	AIX	-
	HP-UX	-
	Solaris	`$(FC) $(FFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTIONS) $<`
	Tru64	`$(FC) $(FFLAGS) -c $<`

	GNU	`$(FC) $(FFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c`
		`$(OUTPUT_OPTION) $<`

	POSIX	-
.F~.o	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.F`
		`$(FC) $(FFLAGS) -c  $*.F`

	Tru64	-
	GNU	-
	POSIX	-
.F.out	AIX	-
	HP-UX	-
	Solaris	-

Suffix	Make	Double suffix rule
	Tru64	`$(FC) $(LDFLAGS) $(FFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-
.ftn.a	AIX	-
	HP-UX	-

	Solaris	`$(F90C) $(F90FLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.ftn~.a	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.ftn`
		`$(F90C) $(F90FLAGS) -c -o $% $*.ftn`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.ftn.o	AIX	-
	HP-UX	-
	Solaris	`$(F90C) $(F90FLAGS) -c $(OUTPUT_OPTION) $<`
	Tru64	-
	GNU	-
	POSIX	-
.ftn.o	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.ftn`
		`$(F90C) $(F90FLAGS) -c $(OUTPUT_OPTION) $*.ftn`

	Tru64	-

Suffix	Make	Double suffix rule
	GNU	-
	POSIX	-
.h~.h	AIX	`$(GET) $(GFLAGS) -p $< > $*.h`
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.h`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.hpp~. hpp	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.hpp`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.hxx~. hxx	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.hxx`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.H~.H	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.H`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-

Suffix	Make	Double suffix rule
.java. class	AIX	-
	HP-UX	-
	Solaris	javac $<
	Tru64	-
	GNU	-
	POSIX	-
.java~. class	AIX	-
	HP-UX	-
	Solaris	$(GET) $(GFLAGS) -p $< > $*.java javac $<
	Tru64	-
	GNU	-
	POSIX	-
.l.c	AIX	$(LEX) $< mv lex.yy.c $@
	HP-UX	$(LEX) $(LFLAGS) $< mv lex.yy.c $@
	Solaris	$(RM) $@ $(LEX) $(LFLAGS) -t $< > $@
	Tru64	$(LEX) $(LFLAGS) $< $(MV) $(MVFLAGS) lex.yy.c $@
	GNU	@$(RM) $@ $(LEX) $(LFLAGS) -t $< > $@
	POSIX	$(LEX) $(LFLAGS) $< mv lex.yy.c $@
.l~.c	AIX	-
	HP-UX	-
	Solaris	$(GET) $(GFLAGS) -p $< > $*.l $(LEX) $(LFLAGS) $*.l mv lex.yy.c $@
	Tru64	-

Suffix	Make	Double suffix rule
	GNU	-
	POSIX	-
.l.ln	AIX	-
	HP-UX	-

	Solaris	`$(RM) $*.c`
		`$(LEX) $(LFLAGS) -t $< > $*.c`
		`$(LINT) $(LINTFLAGS) $(CPPFLAGS) -o $@ -i $*.c`
		`$(RM) $*.c`

	Tru64	-

	GNU	`@$(RM) $*.c`
		`$(LEX) $(LFLAGS) -t $< > $*.c`
		`$(LINT) $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -i $*.c -o $@`
		`$(RM) $*.c`

	POSIX	-
.l~.ln	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.l`
		`$(RM) $*.c`
		`$(LEX) $(LFLAGS) -t $*.l > $*.c`
		`$(LINT) $(LINTFLAGS) $(CPPFLAGS) -o $@ -i $*.c`
		`$(RM) $*.c`

	Tru64	-
	GNU	-
	POSIX	-

.l.o	AIX	`$(LEX) $(LFLAGS) $<`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`rm -f lex.yy.c $*.l`
		`mv lex.yy.c $@`

	HP-UX	`$(LEX) $(LFLAGS) $<`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`rm lex.yy.c`
		`mv lex.yy.o $@`

	Solaris	`$(RM) $*.c`
		`$(LEX) $(LFLAGS) -t $< > $*.c`
		`$(CC) $(CFLAGS) $(CPPFLAGS) -c -o $@ $*.c`
		`$(RM) $*.c`

	Tru64	`$(LEX) $(LFLAGS) $<`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`$(RM) $(RMFLAGS) lex.yy.c`
		`$(MV) $(MVFLAGS) lex.yy.o $@`

Suffix	Make	Double suffix rule
	GNU	-

	POSIX	`$(LEX) $(LFLAGS) <$`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`$rm -f lex.yy.c`
		`mv lex.yy.o $@`

.l~.o	AIX	`$(GET) $(GFLAGS) -p $< > $*.l`
		`$(LEX) $(LFLAGS) $*.l`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`rm -f lex.yy.c $*.l`
		`mv lex.yy.c $*.o`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.l`
		`$(LEX) $(LFLAGS) $*.l`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`rm -f lex.yy.c $*.l`
		`mv lex.yy.o $*.o`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.l`
		`$(LEX) $(LFLAGS) $*.l`
		`$(CC) $(CFLAGS) -c lex.yy.c`
		`rm -f lex.yy.c`
		`mv lex.yy.c $@`

	Tru64	-
	GNU	-
	POSIX	-
.l.out	AIX	-
	HP-UX	-
	Solaris	-

	Tru64	`$(LEX) $(LFLAGS) $<`
		`$(CC) $(LDFLAGS) $(CFLAGS) lex.yy.c $(LOADLIBES) -ll -o $@`
		`$(RM) $(RMFLAGS) lex.yy.c`

	GNU	-
	POSIX	-
.l.r	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-

	GNU	`$(LEX) $(LFLAGS) -t $< > $@`
		`mv -f lex.yy.r $@`

	POSIX	-

Suffix	Make	Double suffix rule
.L.C	AIX	-

	HP-UX	`$(LEX) $(LFLAGS) $<`
		`mv lex.yy.c $@`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.L~.C	AIX	-

	HP-UX	`$(GET) $(GFLAGS) $<`
		`$(LEX) $(LFLAGS) $*.L`
		`mv lex.yy.c $@`
		`-rm -f $*.L`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.L~.L	AIX	-
	HP-UX	`$(GET) $(GFLAGS) $<`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.L.o	AIX	-

	HP-UX	`$(LEX) $(LFLAGS) $<`
		`$(CXX) $(CXXFLAGS) -c lex.yy.c`
		`-rm -f lex.yy.c; mv lex.yy.o $@`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.L~.o	AIX	-

Suffix	Make	Double suffix rule

	HP-UX	`$(GET) $(GFLAGS) $<`
		`$(LEX) $(LFLAGS) $*.L`
		`$(CXX) $(CXXFLAGS) -c lex.yy.c`
		`-rm -f lex.yy.c $*.L`
		`mv lex.yy.o $@`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.mod.a	AIX	-
	HP-UX	-

	Solaris	`$(M2C) $(M2FLAGS) $(MODFLAGS) -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.mod~.a	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.mod`
		`$(M2C) $(M2FLAGS) $(MODFLAGS) -o $% $*.mod`

	Tru64	-
	GNU	-
	POSIX	-
.mod.o	AIX	-
	HP-UX	-
	Solaris	`$(M2C) $(M2FLAGS) $(MODFLAGS) -o $@ $<`
	Tru64	-
	GNU	`$(M2C) $(M2FLAGS) $(MODFLAGS) $(TARGET_ARCH) -o $@ $<`
	POSIX	-
.mod~.o	AIX	-

Suffix	Make	Double suffix rule
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.mod`
		`$(M2C) $(M2FLAGS) $(MODFLAGS) -o $@ $*.mod`

	Tru64	-
	GNU	-
	POSIX	-
.o.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(CC) $(LDFLAGS) $(CFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-
.p.a	AIX	-
	HP-UX	`$(PC) $(PFLAGS) +a -c $<`

	Solaris	`$(PC) $(PFLAGS) $(CPPFLAGS) -c -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.p.a~	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.p`
		`$(PC) $(PFLAGS) +a -c $*.p`
		`-rm -f $*.p`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.p`
		`$(PC) $(PFLAGS) $(CPPFLAGS) -c -o $% $*.p`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.p.o	AIX	-

Suffix	Make	Double suffix rule
	HP-UX	`$(PC) $(PFLAGS) -c $<`
	Solaris	`$(PC) $(PFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $<`
	Tru64	`$(PC) $(PFLAGS) -c $<`
	GNU	-
	POSIX	-
.p~.o	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.p`
		`$(PC) $(PFLAGS) -c $*.p`
		`-rm -f $*.p`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.p`
		`$(PC) $(PFLAGS) $(CPPFLAGS) -c $(OUTPUT_OPTION) $*.p`

	Tru64	-

	GNU	`$(PC) $(PFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c`
		`$(OUTPUT_OPTION) $<`

	POSIX	-
.p.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(PC) $(LDFLAGS) $(PFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-
.p~.p	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.p`
	Solaris	-
	Tru64	-
	GNU	
	POSIX	-
.r.a	AIX	-

	HP-UX	`$(FC) -c $(RFLAGS) $<`
		`ar rv $@ $*.o`
		`rm -f $*.o`

Suffix	Make	Double suffix rule

	Solaris	`$(COMPILE.r) -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.r~.a	AIX	-

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.r`
		`$(FC) -c $(RFLAGS) $*.r`
		`ar rv $@ $*.o`
		`rm -f $*.[ro]`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.r`
		`$(COMPILE.r) -o $% $*.r`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.r.f	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) -F $(OUTPUT_OPTION) $<`
	POSIX	-
.r.o	AIX	-
	HP-UX	`$(FC) $(RFLAGS) -c $<`
	Solaris	`$(COMPILE.r) $(OUTPUT_OPTION) $<`
	Tru64	`$(RC) $(RFLAGS) -c $<`
	GNU	`$(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) -c $(OUTPUT_OPTION) $<`
	POSIX	-
.r~.o	AIX	-

Suffix	Make	Double suffix rule

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.r`
		`$(FC) $(RFLAGS) -c $*.r`
		`-rm -f $*.r`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.r`
		`$(COMPILE.r) $(OUTPUT_OPTION) $*.r`

	Tru64	-
	GNU	-
	POSIX	-
.r.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(RC) $(LDFLAGS) $(RFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-
.r~.r	AIX	-
	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.r`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.s.a	AIX	-
	HP-UX	-

	Solaris	`$(AS) $(ASFLAGS) -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-

.s~.a	AIX	`$(GET) $(GFLAGS) -p $< > $*.s`
		`$(AS) $(ASFLAGS) -o $*.o $*.s`
		`$(AR) $(ARFLAGS) $@ $*.o`
		`-rm -f $*.[so]`

Suffix	Make	Double suffix rule

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.s`
		`$(AS) $(ASFLAGS) -o $*.o $*.s`
		`ar rv $@ $*.o`
		`-rm -f $*.[so]`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.s`
		`$(AS) $(ASFLAGS) -o $% $*.s`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.s.o	AIX	`$(AS) $(ASFLAGS) -o $@ $<`
	HP-UX	`$(AS) $(ASFLAGS) -o $@ $<`
	Solaris	`$(AS) $(ASFLAGS) -o $@ $<`
	Tru64	`$(AS) $(ASFLAGS) -o $@ $<`
	GNU	`$(AS) $(ASFLAGS) $(TARGET_MACH) -o $@ $<`
	POSIX	-

.s~.o	AIX	`$(GET) $(GFLAGS) -p $< > $*.s`
		`$(AS) $(ASFLAGS) -o $*.o $*.s`
		`-rm -f *.s`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.s`
		`$(AS) $(ASFLAGS) -o $*.o $*.s`
		`-rm -f $*.s`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.s`
		`$(AS) $(ASFLAGS) -o $@ $*.s`

	Tru64	-
	GNU	-
	POSIX	-
.s.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	`$(CC) $(LDFLAGS) $(CFLAGS) $< $(LOADLIBES) -o $@`
	GNU	-
	POSIX	-

Suffix	Make	Double suffix rule
.S.a	AIX	-
	HP-UX	-

	Solaris	`$(AS) $(ASFLAGS) -o $% $<`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.S~.a	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.S`
		`$(AS) $(ASFLAGS) -o $% $*.S`
		`$(AR) $(ARFLAGS) $@ $%`
		`$(RM) $%`

	Tru64	-
	GNU	-
	POSIX	-
.S.o	AIX	-
	HP-UX	-
	Solaris	`$(AS) $(ASFLAGS) -o $@ $<`
	Tru64	-
	GNU	`$(CC) $(ASFLAGS) $(CPPFLAGS) $(TARGET_MACH) -c -o $@ $<`
	POSIX	-
.S~.o	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.S`
		`$(AS) $(ASFLAGS) -o $@ $*.S`

	Tru64	-
	GNU	-
	POSIX	-
.S.s	AIX	-

Suffix	Make	Double suffix rule
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(CC) -E $(CPPFLAGS) $< > $@`
	POSIX	-
.tex.dvi	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(TEX) $<`
	POSIX	-
.texi.info	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(MAKEINFO) $(MAKEINFO_FLAGS) $< -o $@`
	POSIX	-
.texinfo. info	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(MAKEINFO) $(MAKEINFO_FLAGS) $< -o $@`
	POSIX	-
.texi.dvi	AIX	-
	HP-UX	-
	Solaris	-

Suffix	Make	Double suffix rule
	Tru64	-
	GNU	`$(TEXI2DVI) $(TEXI2DVI_FLAGS) $<`
	POSIX	-
.texinfo.dvi	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(TEXI2DVI) $(TEXI2DVI_FLAGS) $<`
	POSIX	-
.txinfo.info	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(MAKEINFO) $(MAKEINFO_FLAGS) $< -o $@`
	POSIX	-
.txinfo.dvi	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(TEXI2DVI) $(TEXI2DVI_FLAGS) $<`
	POSIX	-
.w.c	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-

Suffix	Make	Double suffix rule
	GNU	`$(CTANGLE) $< - $@`
	POSIX	-
.w.tex	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(CWEAVE) $< - $@`
	POSIX	-
.web.p	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(TANGLE) $<`
	POSIX	-
.web.tex	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	-
	GNU	`$(WEAVE) $<`
	POSIX	-

.y.c	AIX	`$(YACC) $(YFLAGS) $<`
		`mv y.tab.c $@`

	HP-UX	`$(YACC) $(YFLAGS) $<`
		`mv y.tab.c $@`

	Solaris	`$(YACC) $(YFLAGS) $<`
		`mv y.tab.c $@`

	Tru64	`$(YACC) $(YFLAGS) $<`
		`$(MV) $(MVFLAGS) y.tab.c $@`

	GNU	`$(YACC) $(YFLAGS) $<`
		`mv -f y.tab.c $@`

Suffix	Make	Double suffix rule

	POSIX	`$(YACC) $(YFLAGS) $<`
		`mv y.tab.c $@`

.y~.c	AIX	`$(GET) $(GFLAGS) -p $< > $*.y`
		`$(YACC) $(YFLAGS) $*.y`
		`mv y.tab.c $*.c`
		`-rm -f $*.y`

	HP-UX	`$(GET) $(GFLAGS) -p $< > $*.y`
		`$(YACC) $(YFLAGS) $*.y`
		`mv y.tab.c $*.c`
		`-rm -f $*.y`

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.y`
		`$(YACC) $(YFLAGS) $*.y`
		`mv y.tab.c $@`

	Tru64	-
	GNU	-
	POSIX	-
.y.ln	AIX	-
	HP-UX	-

	Solaris	`$(YACC) $(YFLAGS) $<`
		`$(LINT) $(LINTFLAGS) $(CPPFLAGS) -o $@ -i y.tab.c`
		`$(RM) y.tab.c`

	Tru64	-

	GNU	`$(YACC) $(YFLAGS) $<`
		`$(LINT) $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -C$* y.tab.c`
		`$(RM) y.tab.c`

	POSIX	-
.y~.ln	AIX	-
	HP-UX	-

	Solaris	`$(GET) $(GFLAGS) -p $< > $*.y`
		`$(YACC) $(YFLAGS) $*.y`
		`$(LINT) $(LINTFLAGS) $(CPPFLAGS) -o $@ -i y.tab.c`
		`$(RM) y.tab.c`

	Tru64	-
	GNU	-
	POSIX	-

.y.o	AIX	`$(YACC) $(YFLAGS) $<`
		`$(CC) $(CFLAGS) -c y.tab.c`
		`rm -f y.tab.c`
		`mv y.tab.o $@`

Suffix	Make	Double suffix rule
	HP-UX	$(YACC) $(YFLAGS) $< $(CC) $(CFLAGS) -c y.tab.c rm y.tab.c mv y.tab.o $@
	Solaris	$(YACC) $(YFLAGS) $< $(CC) $(CFLAGS) $(CPPFLAGS) -c -o $@ y.tab.c $(RM) y.tab.c
	Tru64	$(YACC) $(YFLAGS) $< $(CC) $(CFLAGS) -c y.tab.c $(RM) $(RMFLAGS) y.tab.c $(MV) $(MVFLAGS) y.tab.o $@
	GNU	-
	POSIX	$(YACC) $(YFLAGS) $< $(CC) $(CFLAGS) -c y.tab.c rm -f y.tab.c mv y.tab.o $@
.y~.o	AIX	$(GET) $(GFLAGS) -p $< > $*.y $(YACC) $(YFLAGS) $*.y $(CC) $(CFLAGS) -c y.tab.c rm -f y.tab.c $*.y mv y.tab.o $*.o
	HP-UX	$(GET) $(GFLAGS) -p $< > $*.y $(YACC) $(YFLAGS) $*.y $(CC) $(CFLAGS) -c y.tab.c rm -f y.tab.c $*.y mv y.tab.o $*.o
	Solaris	$(GET) $(GFLAGS) -p $< > $*.y $(YACC) $(YFLAGS) $*.y $(CC) $(CFLAGS) -c y.tab.c rm -f y.tab.c mv y.tab.o $@
	Tru64	-
	GNU	-
	POSIX	-
.y.out	AIX	-
	HP-UX	-
	Solaris	-
	Tru64	$(YACC) $(YFLAGS) $< $(CC) $(LDFLAGS) $(CFLAGS) y.tab.c $(LOADLIBES) -ly -o $@ $(RM) $(RMFLAGS) y.tab.c
	GNU	-
	POSIX	-

Suffix	Make	Double suffix rule
.Y.C	AIX	-

	HP-UX	`$(YACC) $(YFLAGS) $<`
		`mv y.tab.c $@`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.Y~.C	AIX	-

	HP-UX	`$(GET) $(GFLAGS) $<`
		`$(YACC) $(YFLAGS) $*.Y`
		`mv y.tab.c $*.C`
		`-rm -f $*.Y`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.Y.o	AIX	-

	HP-UX	`$(YACC) $(YFLAGS) $<`
		`$(CXX) $(CXXFLAGS) -c y.tab.c`
		`-rm -f y.tab.c`
		`mv y.tab.o $@`

	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.Y~.o	AIX	-

	HP-UX	`$(GET) $(GFLAGS) $<`
		`$(YACC) $(YFLAGS) $*.Y`
		`$(CXX) $(CXXFLAGS) -c y.tab.c`
		`-rm -f y.tab.c $*.Y`
		`mv y.tab.o $*.o`

	Solaris	-
	Tru64	-
	GNU	-

Suffix	Make	Double suffix rule
	POSIX	-
.Y~.Y	AIX	-
	HP-UX	`$(GET) $(GFLAGS) $<`
	Solaris	-
	Tru64	-
	GNU	-
	POSIX	-
.ye.e	AIX	-
	HP-UX	-
	Solaris	-

	Tru64	`$(YACCE) $(YFLAGS) $<`
		`$(MV) $(MVFLAGS) y.tab.e $@`

	GNU	-
	POSIX	-
.ye.o	AIX	-
	HP-UX	-
	Solaris	-

	Tru64	`$(YACCE) $(YFLAGS) $<`
		`$(EC) $(EFLAGS) -c y.tab.e`
		`$(RM) $(RMFLAGS) y.tab.e`
		`$(MV) $(MVFLAGS) y.tab.o $@`

	GNU	-
	POSIX	-
.yr.o	AIX	-
	HP-UX	-
	Solaris	-

	Tru64	`$(YACCR) $(YFLAGS) $<`
		`$(RC) $(RFLAGS) -c y.tab.r`
		`$(RM) $(RMFLAGS) y.tab.r`
		`$(MV) $(MVFLAGS) y.tab.o $@`

	GNU	-

Suffix	Make	Double suffix rule
	POSIX	-
.ye.e	AIX	-
	HP-UX	-
	Solaris	-

	Tru64	`$(YACCR) $(YFLAGS) $<`
		`$(MV) $(MVFLAGS) y.tab.r $@`

	GNU	-
	POSIX	-

Appendix C. C compiler options

This appendix contains all the compiler options for the C compilers on AIX, HP-UX, Solaris, and Tru64. These tables let you find the compiler options that you are using on your source platform and then have a look at the corresponding option for the C compiler on AIX.

C.1 Licensing compiler options

Compiler options that are associated with licensing, as shown in Table 112. This is only valid for Solaris.

Table 112. Licensing options

Operating system	Option	Value and default	Description
AIX	n/a	n/a	n/a
HP-UX	n/a	-	-
Solaris	-noqueue	n/a	Instructs the compiler not to queue the compilation if a license is not available.
Solaris	-xlicinfo	-	Returns information about the licensing system.
Tru64	n/a	n/a	n/a

C.2 Standards compliance compiler options

Standard compiler compliance compiler options (shown in Table 113) are compiler options that let your compiler comply with a certain standard.

Table 113. Standards compliance options

Operating system	Option	Value and default	Description
AIX	-qgenproto	"" or parmnames; default is nogenproto	Produces ANSI prototypes from K&R function definitions.
AIX	-qnogenproto		Does not produce ANSI prototypes from K&R function definitions.

Operating system	Option	Value and default	Description
AIX	-qlanglvl	ANSI or SAAL2, SAA, EXTended, CLAssic, NOUCS, or UCS; default is ANSI with xlc or c89, extended when using cc	Selects the C language level for compilation.
AIX	-qlibansi		Assumes that all functions with the name of an ANSI C library function are in fact the system functions.
AIX	-qnolibansi		Turns off the assumption that all functions with the name of an ANSI C library function are in fact the system functions.
HP-UX	-Aa		Enables strict ANSI C compliance.
HP-UX	-Ac		Disables ANSI C compliance (HP C Version 3.1 compatibility).
HP-UX	-Ae		Enables ANSI C compliance, HP value-added features (as described for +e option), and _HPUX_SOURCE name space macro. It is equivalent to -Aa +e -D_HPUX_SOURCE.
HP-UX	+e		Enables the following HP value added features while compiling in ANSI C mode: sized enum, long long, long pointers, compiler supplied defaults for missing arguments to intrinsic calls, and $ in identifier HP C extensions.
Solaris	-X	a, c, s, or t	The -X options specify varying degrees of compliance to the ANSI/ISO C standard.
Tru64	-std		Selects the relaxed ANSI language mode. This is the default. Enforces the ANSI C standard, but allows some common programming practices disallowed by the standard.

Operating system	Option	Value and default	Description
Tru64	-std0		Selects the K & R language mode. Enforces the K & R programming style, with certain ANSI extensions in areas where the K & R behavior is undefined or ambiguous. In general, -std0 compiles most pre-ANSI C programs and produces expected results. The -std0 option causes the __STDC__ macro to be undefined.
Tru64	-std1		Selects the strict ANSI language mode. Strictly enforces the ANSI C standard and all its prohibitions (such as those that apply to the handling of void types, the definition of lvalues in expressions, the mixing of integrals and pointers, and the modification of rvalues).
Tru64	-ms		Selects the Microsoft language mode, which provides some compatibility with the Microsoft Visual C compiler. Although this option does not provide full compatibility, it can be useful as a porting aid.
Tru64	-common		Selects the K & R language mode. This option is equivalent to -std0.
Tru64	-traditional		Selects the K & R language mode. This option is equivalent to -std0.
Tru64	-vaxc		Selects the VAX C language mode. This is similar to -std (relaxed ANSI mode), but extends the language semantics in ways that are incompatible with ANSI C. It provides close compatibility with the vaxc compiler.
Tru64	-isoc94		Causes the macro __STDC_VERSION__ to be passed to the preprocessor and enables recognition of the digraph forms of various operators. Note that the -isoc94 option has no influence on -stdn options and vice versa.

C.3 Optimization and performance compiler options

Modern compilers apply several different methods to optimize your program. We have tried to group the various methods together in tables.

C.3.1 Aliasing

Table 114 shows the aliasing options.

Table 114. Aliasing options

Operating system	Option	Value and default	Description
AIX	-qalias	TYPeptr, [NO]TYPeptr, ALLPtrs, [NO]ALLPtrs, [ADDRtaken, [NO]ADDRtaken, ANSI, or [NO]ANSI	Specifies whether type-based aliasing is to be used during optimization. If used, `#pragma ALIAS=suboption` must appear before the first program statement.
AIX	-qansialias		Specifies that type-based aliasing is to be used during optimization. This option is obsolete. Use -qalias= in your new applications.
AIX	-qnoansialias	default	Specifies that no type-based aliasing is to be used during optimization. This option is obsolete. Use -qalias= in your new applications.
AIX	-qassert	TYPeptr, ALLPtrs, ADDRtaken, or noassert	Requests the compiler to apply aliasing assertions to your compilation unit. This option is obsolete and -qalias should be used.
Tru64	-ansi_alias		Directs the compiler to assume the ANSI C aliasing rules, and thus allows the optimizer to be more aggressive in its optimizations.
Tru64	-noansi_alias		Directs the compiler not to assume the ANSI C aliasing rules, and thus not allow the optimizer to be more aggressive in its optimizations.

C.3.2 Inlining

Table 115 shows the inlining options.

Table 115. Inlining options

Operating system	Option	Value and default	Description
AIX	-qinline	See manual	Attempts to inline functions instead of generating calls to a function.
AIX	-qnoinline	See manual	Disables attempts to inline functions.
AIX	-Q		Attempts to inline functions instead of generating calls to a function. The -Q option is equivalent to the -qinline function.
AIX	-ma		Substitutes inline code for calls to function alloca as if `#pragma alloca` directives are in the source code.
HP-UX	+ESfic		Compiles with inline fast indirect calls.
HP-UX	+ESsfc		Replaces function pointer comparison millicode calls with inline code.
Solaris	-xinline		Tries to inline only those functions specified.
Solaris	-xlibmil		Inlines some library routines for faster execution.
Solaris	-xnolibmil		Does not inline math library routines.
Solaris	-xcrossfile=	0 or 1; default is 1	Enables optimization and inlining across source files (SPARC only).
Tru64	-inline	none, manual, size, speed, or all	Specifies whether to provide inline expansion of functions.
Tru64	-noinline		No inline optimization.
Tru64	-preempt_ module		Supports symbol preemption on a module-by-module basis. During optimization, inlining is performed only on functions within a single compilation unit.

C.3.3 Side effects

Table 116 shows the side effects options.

Table 116. Side effects options

Operating system	Option	Value and default	Description
AIX	-qignerrno		Allows the compiler to perform optimizations that assume errno is not modified by system calls.
AIX	-qnoignerrno		Tells the compiler not to perform optimizations that assume errno is not modified by system calls.
AIX	-qisolated_ call=	Function	Specifies functions in the source file that have no side effects.

C.3.4 Code size reduction

Table 117 shows the code size reduction options.

Table 117. Code size reduction options

Operating system	Option	Value and default	Description
AIX	-qcompact	nocampct	When used with optimization, reduces code size where possible, at the expense of execution speed.
AIX	-qnocompact	Default	When used with optimization, does not reduce code size.
AIX	-qonce		Avoids including a header file more than once, even if it is specified in several of the files you are compiling.
AIX	-qnoonce	Default	Do not avoid including a header file more than once, even if it is specified in several of the files you are compiling.
Tru64	-compress		Passes the -compress option to the compilation phase (if the -c option is present) or passes the -compress_r option to `ld` (if the -r option is present). Use of this option causes the output object file to be produced in compressed object file format, resulting in a substantially smaller object file.
Tru64	-om_dead_cod e		Removes dead code (unreachable instructions) generated after applying optimizations. The .lita section is not compressed by this option.
Tru64	-om_compress _lita		Removes unused .lita entries after optimization, and then compresses the .lita section.

C.3.5 Compile time optimization

Table 118 shows the compile time optimization options.

Table 118. Compile time optimization options

Operating system	Option	Value and default	Description
AIX	-qgenpcomp	nopcomp	Generates a precompiled version of any header file for which the original source is used.
AIX	-qmaxmem	2048 KB, -1 for unlimited.	Limits the amount of memory used for local tables of specific, memory-intensive optimizations.
AIX	-qnopcomp		Disables generation of a precompiled version of any header file for which the original source is used.
AIX	-qspill	Default is 512	Specifies the size of the register allocation spill area.

C.3.6 Performance data collection

Table 119 shows the performance data collection options.

Table 119. Performance data collection options

Operating system	Option	Value and default	Description
AIX	-qfdpr	nofdpr	Collects program information for use with the AIX fdpr performance-tuning utility.
AIX	-qnofdpr		Disables collection of program information for use with the AIX fdpr performance-tuning utility.
AIX	-qpdf1	See manual	Tunes optimizations through Profile-Directed Feedback. For optimum performance, use the -O3 option with all compilations when you use PDF.
AIX	-qnopdf1	Default	Turns off tuned optimizations through Profile-Directed Feedback.
AIX	-qpdf2	See manual	Tunes optimizations through Profile-Directed Feedback.
AIX	-noqpdf2	Default	Turns off tuned optimizations through Profile-Directed Feedback.
AIX	-p		Sets up the object files produced by the compiler for profiling.

Operating system	Option	Value and default	Description
AIX	-pg		Sets up the object files produced by the compiler for profiling. Provides more information than is provided by the -p option.
AIX	-profile=	ibm, p, or, pg	Sets up the object files produced by the compiler for profiling. The suboption indicates the profiling tool. If the -qtbtable option is not set, the -qprofile option will generate full traceback tables.
AIX	-qnoproile		Disables profiling.
HP-UX	+dfname		Specifies the profile database to use with profile-based optimization.
HP-UX	+P		Performs profile-based optimization.
HP-UX	+pa		Requests routine-level profiling with CXperf.
HP-UX	+pal		Requests routine-level and loop-level profiling with CXperf.
HP-UX	+pgmname		Specifies the execution profile data set to be used by the optimizer.
HP-UX	-y		Generates information used by the HP SoftBench static analysis tool.
HP-UX	-G		Inserts information required by the gprof profiler in the object file.
HP-UX	-p		Inserts information required by the prof profiler in the object file.
HP-UX	+I		Prepares the object code for profile-based optimization data collection.
Solaris	-xF		Enables performance analysis of the executable using the analyzer.
Solaris	-p		Prepares the object code to collect data for profiling.
Solaris	-xa		Inserts code to count how many times each basic block is executed.
Solaris	-xprofile=	collect[X], use[X], or tcov; default for X is a.out	Collects data for a profile or uses a profile to optimize (SPARC only).

Operating system	Option	Value and default	Description
Tru64	-feedback	File	Specifies that the compiler should use the profile information contained in file when performing optimizations.
Tru64	-om_split_ procedures		Splits frequently accessed routines into "hot" and "cold" code segments, and stores these segments in different parts of the image. The hot segments are the most frequently executed parts of the code, as determined by feedback data produced by a representative run of the program. The hot segments are stored near other parts of the program that are also executed frequently. In this way, the most frequently executed parts of the program are compacted in a way that makes them more likely to fit into the cache. This speeds up the execution time of that code.
Tru64	-om_feedback		Uses the pixie-produced information stored in the augmented executable by means of the `cc` command's -feedback option and the `pixie` (or `prof`) command's -update option.
Tru64	-om_ireorg_ feedback, file	file	Uses the `pixie`-produced information in file. Counts and file.Addrs to reorganize the instructions to reduce cache thrashing.
Tru64	-spike		Invokes the spike tool to perform code optimization after linking a program. -spike is a replacement for -om and does similar optimizations. See spike(1) for complete information on the `spike` command, its options, and its relationship to -spike.
Tru64	-gen_feedback		Generates accurate profile information to be used with -feedback optimizations.
Tru64	-prof_gen		Generates an executable image that has profiling code added to it. Using this option is equivalent to running the `pixie` command on an existing image. The `pixie`-instrumented file is called a.out (or as specified with the -o option).
Tru64	-prof_gen_noopt		Generates a non-optimized executable image that, when run, will generate profiling data that the compiler can use to improve its optimization choices (with -prof_use_feedback).

Operating system	Option	Value and default	Description
Tru64	-prof_use_ feedback		Uses profiling feedback to improve compiler optimization. Using this option is equivalent to using the `prof` command to produce a feedback file, and then using the `cc -feedback` command to recompile the program.
Tru64	-prof_use_om_ feedback		Uses profiling feedback to rearrange the resulting image to reduce cache conflicts of the program text. This option uses the -om postlink optimizer, and is equivalent to using the -om -WL, -om_ireorg_feedback options. If the -pids option is also specified, this option merges the .Counts performance data files using the `prof -pixie -merge` command.
Tru64	-prof_dir		Specifies a location to which the profiling data files (.Counts and .Addrs) are written. Use this option in conjunction with the -prof_gen option and the -prof_use_feedback or -prof_use_om_feedback option to specify a location for the profiling data files. If you do not specify this option, the profiling files are written to the current directory. Specifying the -prof_dir option also enables the -pids option.
Tru64	-pids		Enables the addition of the process ID to the file name of the basic block counts file (.Counts). This facilitates collecting information from multiple invocations of the `pixie` output file. Unless the -prof_dir option is specified, the default is -nopids.
Tru64	-nopids		Disables the addition of the process ID to the file name of the basic block counts file.
Tru64	-pg		Turns gprof profiling on or off when compiling and linking the file immediately following this option. The gprof profiler produces a call graph showing the execution of a C program.
Tru64	-nopg		Turns off gprof profiling.

C.3.7 Loop optimization

Table 120 shows the loop optimization options.

Table 120. Loop optimization options

Operating system	**Option**	**Value and default**	**Description**
AIX	-qunroll=	0, 1, 2, 3, 4, 5, 6, 7, or 8; default is 4	Unrolls inner loops in the program by a specified factor.
AIX	-qnounroll		Disables unrolling of inner loops in the program.
AIX	-qstrict_induction	Default when using -O level 0 or using c89	Disables loop induction variable optimizations that have the potential to alter the semantics of your program.
AIX	-qnostrict_induction	Otherwise	Allows loop induction variable optimizations that have the potential to alter the semantics of your program.
Solaris	-xdepend		Analyzes loops for inter-iteration data dependencies and does loop restructuring (SPARC only).
Solaris	-xspace		Does no optimizations or parallelization of loops that increase code size.
Solaris	-xunroll=	1-N	Suggests, to the optimizer, to unroll loops n times.
Tru64	-unroll n	0-N	Controls the loop-unrolling optimization (available only at levels -O2 and higher). -unroll n allows the compiler to unroll loops up to n times. -unroll 1 disables the optimization. -unroll 0 (the default) allows the compiler to decide what is best.

C.3.8 Processor and architectural optimization

Table 121 shows the processor and architectural options.

Table 121. Processor and architectural options

Operating system	Option	Value and default	Description
AIX	-qcache=	assoc=number, auto, cost=cycles, level=level, line=bytes, size=Kbytes, type=cache_type	Specifies the cache configuration for a specific execution machine. The -qcache option has an effect only if you also specify the -qipa, -O4, -O5, or -qsmp options.
Solaris	-xcache=	generic, s1/l1/a1, s1/l1/a1:s2/l2/a2, or s1/l1/a1:s2/l2/a2:s3/l3/a3	Defines the cache properties for use by the optimizer.
AIX	-qtune=	auto, 403, 601, 603, 604, p2sc, pwr2, pwr3, pwrx, rs64a, or rs64b	Specifies the architecture for which the executable program is optimized. (If -qtune is specified without -qarch=suboption, the compiler uses -qarch=com.)
HP-UX	+DS	model	Performs instruction scheduling for a specific implementation of PA-RISC.
Solaris	-xchip=	generic, old, super, super2, micro, micro2, hyper, hyper2, powerup, ultra, ultra2, ultra3, ultra2i, 386, 486, pentium, pentium_pro, 603, or 604	Specifies the target processor for use by the optimizer. The documentation is a bit unclear whether PPC603 and PPC604 is supported
Solaris	-x386		Optimizes for the 80386 processor.
Solaris	-x486		Optimizes for the 80486 processor.
Solaris	-xpentium		Optimizes for the Pentium processor.
Solaris	-xtarget		Specifies the target system for instruction set and optimization.

Operating system	Option	Value and default	Description
Tru64	-tune	generic, host, ev4, ev5, ev56, ev6, or ev67	Instructs the optimizer to tune the application for a specific version of the Alpha hardware. This will not prevent the application from running correctly on other versions of Alpha, but it may run more slowly than generically-tuned code on those versions.

C.3.9 Optimization spreading across several files

Table 122 shows the multiple file optimization options.

Table 122. Multiple file optimization

Operating system	Option	Value and default	Description
AIX	-qipa=	See manual.	Turns on or customizes a class of optimizations known as interprocedural analysis (IPA). If the -S compiler option is specified with noobject, noobject is ignored.
Tru64	-ifo		Provides improved optimization (inter-file optimization) and code generation across file boundaries that would not be possible if the files were compiled separately.

C.3.10 Optimization flags (-O and family)

Table 123 shows the optimization flags.

Table 123. Optimization flags

Operating system	Option	Value and default	Description
AIX	-O	"", 2, 3, 4, or 5	Optimizes code at a choice of levels during compilation.
AIX	-qOPTimize	Same as -O	Optimizes code at a choice of levels during compilation.
HP-UX	-fast		Expands into a set of compiler options, which results in improved application run-time. Options included are +O3, +Onolooptransform, +Olibcalls, +FPD, +Oentryschedule, and +Ofastaccess. Any of these options can be overridden by placing a subsequent option after -fast on the command line.
HP-UX	-O		Optimizes at level 2.

Operating system	Option	Value and default	Description
HP-UX	+Oopt	opt or opt=func, ffunc	Invokes optimization level opt where opt is 0 to 4. See the HP C/HP-UX Programmer's Guide for additional optimization options.
HP-UX	-B	extern	Performs the same operation as +Oextern=sym1,sym2,sym3..., except that symbols are loaded from an existing file instead of specified on the command line.
Solaris	-fast		Selects the optimum combination of compilation options for speed.
Tru64	-fast		Provides a single method for turning on a collection of optimizations for increased performance.Note that the -fast option can produce different results for floating- point arithmetic and math functions, although most programs are not sensitive to these differences.
Solaris	-xo	1, 2, 3, 4, or 5	Optimizes the object code.
Solaris	-O		The same as -xo2.
Tru64	-O	1, 2, 3, or 4	Determines the level of optimization.

C.3.11 Limiting of optimization options

Table 124 shows the restricting optimization options.

Table 124. Restricting optimization

Operating system	Option	Value and default	Description
AIX	-qstrict	Default depends on -Olevel.	Turns off aggressive optimizations that have the potential to alter the semantics of your program.
AIX	-qnostrict	Default depends on -Olevel.	Ignores turning off aggressive optimizations that have the potential to alter the semantics of your program.
HP-UX	+ESnoparmreloc		Disables parameter relocation for function calls.

Operating system	Option	Value and default	Description
Solaris	-xmaxopt=	off, 1, 2, 3, 4, or 5. Default is off.	This command limits the level of `pragma` opt to the level specified.
Tru64	-check_omp		Enables run-time checking of certain OpenMP constructs. This includes run-time detection of invalid nesting and other invalid OpenMP cases. When invalid nesting is discovered at run time and this option is set, the executable will fail with a Trace/BPT trap.

C.3.12 Other optimization options

Table 125 shows the other optimization flags.

Table 125. Other optimization flags

Operating system	Option	Value and default	Description
AIX	-qgrrcopy	overlap or nooverlap. See the manual for the default.	Enables destructive copy operations for structures and unions (for faster code).
HP-UX	+k		Generates long-displacement code sequences so a program can reference large amounts of global data physically located in shared libraries.
HP-UX	+hugesize		Lowers the threshold for huge data.
Solaris	-xliclib=sunperf		Links in the Sun-supplied performance libraries. The documentation is inconsistent whether this is -xliclib or -xlic_lib.
Solaris	-xprefetch	auto, no%auto, explicit, no%explicit, yes, or no.	Enable prefetch instructions (SPARC only).
Solaris	-xrestrict=	%all, %none, or function name.	Treats pointer-valued function parameters as restricted pointers (SPARC only).
Solaris	-xsafe=mem		Allows the compiler to assume no memory-based traps occur (SPARC only).

Operating system	Option	Value and default	Description
Tru64	-asume	See manual.	With the command, you tell the compiler that it can assume various things. See the man page for cc for more information.
Tru64	-cord		Runs the procedure rearranger, cord, on the resulting file after linking. The rearrangement is done to reduce the cache conflicts associated with accessing the program's text
Tru64	-intrinsics		The -intrinsics option causes the compiler to automatically recognize intrinsic functions wherever it can, based only on name and call signature. Unlike -D_INTRINSICS, this option can treat library function calls as intrinsic even when the appropriate header file is not included. Any function declaration or call site (in the case of implicit declaration), with a name matching the name of an intrinsic function, is examined to see if its parameters and return result are consistent with the intrinsic function of that name. If so, calls are treated as being intrinsic. If not, a diagnostic is issued and calls are treated as ordinary external function calls.
Tru64	-nointrinsics		Disables intrinsics.
Tru64	om_no_inst_sched		Turns off instruction scheduling.
Tru64	om_no_align_labels		Turns off alignment of labels. Normally, the -om option will align the targets of all branches on quadword boundaries to improve loop performance.
Tru64	-ansi_args		Tells the compiler that the source code follows all ANSI rules about arguments, that is, whether the type of an argument matches the type of the parameter in the called function or whether a function prototype is present so the compiler can automatically perform the expected type conversion.

Operating system	Option	Value and default	Description
Tru64	-om_Gcommon, num	common or num	Sets the size threshold of common symbols. Every common symbol whose size is less than or equal to num will be allocated close to each other. This option can be used to improve the probability that the symbol can be accessed directly from the $gp register. (Normally, om tries to collect all common symbols together, not just symbols that conform to certain size constraints.)
Tru64	-[no]ansi_args		Tells the compiler the source code does not follow ANSI rules.
Tru64	-preempt_symbol		Preserves full symbol preemption; that is, supports symbol preemption on a symbol-by-symbol basis within a module as well as between modules. Restricts the optimizer so that calls to extern functions are ineligible for inline replacement.
Tru64	-speculate all		Speculation is a compiler optimization that causes the hardware to begin executing an operation before it determines that the flow of control will actually reach that operation. If the flow of control does reach that operation, the result will be available sooner than it would have been without speculative execution. If the flow of control does not reach that operation, the result will simply be ignored.
Tru64	-speculate by_routine		A module compiled with -speculate by_routine cannot use any form of local exception handling, but can be linked with other modules that do. The run-time system checks each exception to see if it occurred in a speculative routine. It dismisses exceptions from routines that are speculatively executed, but signals exceptions from other routines.

C.4 Data alignment compiler options

Table 126 lists the data alignment compiler options.

Table 126. Data alignment compiler options

Operating system	Option	Value and default	Description
AIX	-qalign	power, full, mac68k, twobyte, packed, bit_packed, or natural Default is full.	Specifies what aggregate alignment rules the compiler uses for compilation. You can code `#pragma options align=reset` in a source file to change the alignment option to what it was before the last alignment option was specified. If no previous alignment rule appears in the file, the alignment rule specified in the invocation command is used.
AIX	-qenum=	small, int, intlong, 1, 2, 4, 8, or RESET. Defaults is int.	Specifies the amount of storage occupied by the enumerations.
HP-UX	+u	Bytes.	Controls pointer alignment where bytes is 1, 2, or 4.
HP-UX	-Y		Enables Native Language Support (NLS).
Solaris	-xchar_byte_order=	low, high, or default.	Produces an integer constant by placing the characters of a multi-character character-constant in the specified byte order. The default places the characters of a multi-character character-constant in an order determined by the compilation mode -X[a|c|s|t].
Solaris	-xmemalign=	1, 2, 4, 8, or 16, followed by i, s, or f.	Specifies maximum assumed memory alignment and behavior of misaligned data accesses.
Tru64	-granularity	Size	Controls the size of shared data in memory that can be safely accessed from different threads. The possible size values are byte, longword, and quadword.
Tru64	-protect_headers	all, none, or default.	Ensures that the compiler's assumptions about pointer sizes and data alignments are not in conflict with the default values that were in effect when the system libraries were created.

Operating system	Option	Value and default	Description
Tru64	-misalign		Lets the compiler assume that misalignment is present in the program.
Tru64	-nomisalign		Lets the compiler assume that no misalignment is present in the program. (This is a synonym for the -asume aligned_objects.)
Tru64	-member_alignment		Directs the compiler to byte-align data structure members (with the exception of bit-field members). By default, data structure members are aligned on natural boundaries instead of the next byte.
Tru64	-nomember_alignment		Directs the compile not to byte-align data structure members.
Tru64	-Zp	1, 2, 4, or 8.	Aligns structure members and entire structures based on the integer n. This option sets a limit on the alignment given to structure members so that each member after the first is stored on a maximum of an n-byte boundary.
Tru64	-strong_volatile		Affects the generation of code for assignments to objects that are less than or equal to 16 bits in size that have been declared as volatile. The generated code includes a load-locked instruction for the enclosing longword or quadword, an insertion of the new value of the object, and a store-conditional instruction for the enclosing longword or quadword.
Tru64	-weak_volatile	Default.	The -weak_volatile option does not generate locked instructions for this sequence. This allows byte or word access to memory-like I/O devices for which larger accesses will not cause read or write side effects. Because the sequence does not access byte or word data independently directly in memory (that is, ensure byte granularity), adjacent volatile data can be corrupted when such byte or word accesses are performed in a multithreaded environment.

C.5 Floating point and numeric compiler options

Here we have compiler options that deal with floating point and other numerical features of the compilers.

C.5.1 Sizes

Table 127 shows the floating point size options.

Table 127. Floating point size

Operating system	Option	Value and default	Description
AIX	-qldbl128		Increases the size of long double type from 64 bits to 128 bits.
AIX	-qnoldbl128		Disables 128 bits long doubles.
AIX	-qldbl128		Increases the size of long double type from 64 bits to 128 bits. The -qlongdouble option is the same as the -qldbl128 option.
AIX	-qnoldbl128		Disables 128 bits long doubles. The -qnolongdouble option is the same as the -qnoldbl128 option.
AIX	-qlonglit		Changes implicit type selection in 64-bit mode to use larger data types where possible.
AIX	-qnolonglit		Does not change implicit type selection in 64-bit mode to use larger data types where possible.
AIX	-qlonglong	Default is cc.	Allows long long types in your program.
AIX	-qnolonglong	Default is c89.	Does not allow long long types in your program.
HP-UX	+f		Inhibits the promotion of float to double, except for function calls and returns.
HP-UX	+r		Inhibits the automatic promotion of float to double.
Tru64	-double		Promotes expressions of type float to double. This is the default when -std0 is used.
Tru64	-float		Prevents the compiler from promoting expressions of type float to type double. This is the default except in -std0 mode.

C.5.2 Rounding of floating points

Table 128 shows the rounding of floating points options.

Table 128. Rounding of floating points options

Operating system	**Option**	**Value and default**	**Description**
AIX	-qrrm		Prevents floating-point optimizations that are incompatible with run-time rounding to plus and minus infinity modes.This option is obsolete. Use -qfloat=rrm in your new applications.
AIX	-qnorrm	default	Does not prevents floating-point optimizations that are incompatible with run-time rounding to plus and minus infinity modes.This option is obsolete. Use -qfloat=rrm in your new applications.
AIX	-y=	n, m, p, or z	Specifies the compile-time rounding mode of constant floating-point expressions.
Solaris	-fprecision=	single, double, or extended	Initializes the rounding-precision mode bits in the floating-point control word. Note that on Intel, only the precision, not exponent, range is affected by the setting of floating-point rounding precision mode.
Solaris	-fround=	nearest, tozero, negative, or positive; default is nearest	Sets the IEEE 754 rounding mode that is established at run time during the program initialization
Tru64	-fprm	c, d, n, or m	Specifies rounding mode for floating points numbers.

C.5.3 Traps

Table 129 shows the floating point traps.

Table 129. Floating point traps

Operating system	Option	Value and default	Description
AIX	-qflttrap	OVerflow, UNDerflow, ZEROdivide, INValid, INEXact, ENable, or IMPrecise	Generates extra instructions to detect and trap floating point exceptions. If specified with `#pragma` options, the -qnoflttrap option must be the first option specified.
HP-UX	+FP	flags	Controls floating-point traps.
Tru64	-fptm	n or u	Generates instructions that do or do not trigger floating-point underflow or inexact trapping modes. Any floating point overflow, divide-by-zero, or invalid operation will unconditionally generate a trap. The -fptm n option is the default.
Tru64	-scope_safe		Ensures that any trap (such as floating-point overflow) is reported to have occurred in the procedure or guarded scope that caused the trap. Any trap occurring outside that scope is not reported to have occurred in the procedure or guarded scope, with the exception of well-defined trapb instructions following jsr instructions.

C.5.4 Single precision

Table 130 shows the single precision options.

Table 130. Single precision options

Operating system	Option	Value and default	Description
AIX	-qrndsngl		Specifies that the result of each single-precision (float) operation is to be rounded to single precision.
AIX	-qnorndsngl	Default	Specifies that the result of each single-precision (float) operation is not to be rounded to single precision.
AIX	-hsflt		Speeds up calculations by removing range checking on single-precision float results and on conversions from floating point to integer.

Operating system	Option	Value and default	Description
AIX	-nohsflt		Enables range checking on single-precision float results and on conversions from floating point to integer
AIX	-hssngl		Specifies that single-precision expressions are rounded only when the results are stored into float memory locations. This option is obsolete. Use -qfloat=hssngl in your new applications.
AIX	-nohssngl		Specifies that single-precision expressions are rounded after expression evaluation. This option is obsolete. Use -qfloat=hssngl in your new applications.
Solaris	-xsfpconst		Represents unsuffixed floating-point constants as single precision.
Solaris	-fsingle		Causes the compiler to evaluate float expressions as single precision rather than double precision.
Tru64	-float_const		Causes the compiler to assign the type float (rather than double) to floating-point constants, if their values can be represented in single precision. This option is not available in -std1 mode.

C.5.5 Other options

Table 131 shows the other floating point options.

Table 131. Other floating point options

Operating system	**Option**	**Value and default**	**Description**
AIX	-qfloat	(no)emulate, (no)fltint, (no)fold, (no)hsflt, (no)hssngl, (no)maf, (no)rndsngl, (no)rrm, (no)rsqrt, or (no)spnans. Default is noemulate, nofltint, fold nohsflt, nohssngl, maf, norndsngl norrm, norsqrt, or nospnans.	Specifies various floating point options to speed up or improve the accuracy of floating point operations.
AIX	-fold		Specifies that constant floating point expressions are to be evaluated at compile time.
AIX	-nofold		Specifies that constant floating point expressions are not to be evaluated at compile time.
AIX	-qmaf		Specifies that the floating-point multiply-add instructions are to be generated. This option is obsolete. Use -qfloat=maf in your new applications.
AIX	-qnomaf		Specifies that the floating-point multiply-add instructions are *not* to be generated. This option is obsolete. Use -qfloat=maf in your new applications.
Solaris	-fnonstd		Causes nonstandard initialization of floating-point arithmetic hardware (SPARC only).
Solaris	-fns=	yes or no.	Turns on the SPARC nonstandard floating-point mode (SPARC only).

Operating system	Option	Value and default	Description
Solaris	-fsimple=n	0, 1, or 2. Default is 0 for no -fsimple, 1 otherwise.	Allows the mizer to make simplifying assumptions concerning floating-point arithmetic.
Solaris	-fstore		Causes the compiler to convert the value of a floating-point expression or function to the type on the left-hand side of an assignment.
Solaris	-ftrap=	%all, %none, common, invalid, no%Invalid, overflow, no%overflow, underflow, no%underflow, division, no%division, inexact, or no%inexact.	Sets the IEEE 754 trapping mode in effect at startup.
Solaris	-nofstore		Does not convert the value of a floating-point expression or function to the type on the left-hand side of an assignment.
Solaris	-xlibmieee		Forces IEEE 754 style return values for math routines in exceptional cases.
Solaris	-xvector=	yes or no. Default is no.	Enable automatic generation of calls to the vector library functions.
Tru64	-fp_reorder		Specifies that code transformations that affect floating- point operations are allowed. These changes can affect the accuracy of the program's results.
Tru64	-ieee		Ensures support of all portable features of the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), including the treatment of denormalized numbers, NaNs, and infinities and the handling of error cases. This option also sets the _IEEE_FP C preprocessor macro.
AIX	-nofp_reorder		Does not allow code transformations that affect floating-point operations. These changes can affect the accuracy of the program's results.

Operating system	Option	Value and default	Description
Tru64	-trapuv		Forces all uninitialized stack variables to be initialized with 0xfff58005fff58005. When this value is used as a floating-point variable, it is treated as a floating-point NaN and causes a floating- point trap. When it is used as a pointer, an address or segmentation violation usually occurs.

C.6 Parallelization compiler options

Table 132 shows the compiler options that deals with parallelization.

Table 132. Parallelization options

Operating system	Option	Value and default	Description
AIX	-qsmp=	See manual	Specifies if and how parallelized object code is generated.
AIX	-qnosmp	default	Specifies that parallelized object code is not to be generated.
AIX	-qthreaded		Indicates to the compiler that the program will run in a multithreaded environment.
AIX	-qnothreaded		Indicates to the compiler that the program will *not* run in a multithreaded environment.
Solaris	-mt		Macro on that expands to -D_REENTRANT -lthread.
Solaris	-xautopar		Turns on automatic parallelization for multiple processors (SPARC only).
Solaris	-xexplicitpar		Generates parallelized code based on specification of `#pragma` MP directives (SPARC only).
Solaris	-xloopinfo		Shows which loops are parallelized and which are not (SPARC only).
Solaris	-xparallel		Shows which loops are parallelized and which are not.
Solaris	-xreduction		Turns on reduction recognition during automatic parallelization (SPARC only).
Solaris	-xvpara		Warns about loops that have `#pragma` MP directives specified but may not be properly specified for parallelization (SPARC only).

Operating system	Option	Value and default	Description
Solaris	-Zll		Creates the program database for lock_lint, but does not actually compile (SPARC only).
Solaris	-Zlp		Prepares object files for the loop profiler (looptool) (SPARC only).
Tru64	-pthread		Directs the linker to use the threadsafe version of any library specified with the -l option when linking programs. This option also tells the linker to include the POSIX 1003.1c-conformant DECthreads interfaces in libpthread when linking the program.
Tru64	-threads		Directs the linker to use the threadsafe version of any library specified with the -l option when linking programs. This option also tells the linker to include the POSIX 1003.4a Draft 4 conformant DECthreads interfaces. It is supported only for compatibility with earlier releases of Tru64 UNIX. New designs should use the -pthread option.
Tru64	-mp		Causes the compiler to recognize an older form of parallel programming directives, as well as OpenMP directives, and pass libots3 and appropriate thread libraries to the linker. It also predefines the macro _OPENMP with a value of 0.
Tru64	-omp		Causes the compiler to recognize the OpenMP shared memory parallel programming API pragmas and pass libots3 and appropriate thread libraries to the linker. It also predefines the macro _OPENMP with a nonzero value.

C.7 Source Code compiler options

Table 133 shows the compiler options that deal with the source code.

Table 133. Source code options

Operating system	Option	Value and default	Description
AIX	-qattr=	all or ““.	Produces a compiler listing that includes an attribute listing for all identifiers.
AIX	-qnoattr	Default.	Does not produces a compiler listing that includes an attribute listing for all identifiers.

Operating system	Option	Value and default	Description
AIX	-B		Determines substitute path names for the compiler, assembler, linkage editor, and preprocessor.
AIX	-C	-B -Bprefix or -B -tprogram -Bprefix -tprograms.	Preserves comments in preprocessed output.The optional prefix defines part of a path name to the new programs. It must end in /.
Tru64	-C		Passes all comments directly to the preprocessor output, except comments on preprocessor directive lines.
AIX	-qcpluscmt		Use this option if you want C++ comments to be recognized in C source files.
AIX	-qnocpluscmt	Default.	Use this option if you *not* want C++ comments to be recognized in C source files.
AIX	-D	Name.	Defines the identifier name as in a `#define` preprocessor directive. -Dname is equal to -Dname=1.
AIX	-qdbcs	Default is nodbcs.	Use the -qdbcs option if your program contains multibyte characters.
AIX	-qnodbcs		Use the -qnodbcs option if your program contains multibyte characters.
AIX	-qdigraph		Allows use of digraph character sequences in your program.
AIX	-qnodigraph	Default.	Does not allow you to use digraph character sequences in your program.
AIX	-E		Runs the source files named in the compiler invocation through the preprocessor.
Tru64	-E		Runs only the C macro preprocessor on the files and sends the result to the standard output device.
AIX	-I	Directory.	Specifies an additional search path if the file name in the `#include` directive is not specified using its absolute path name.

Operating system	Option	Value and default	Description
Tru64	-I	Directory.	Specifies a search path for header files whose names do not indicate a specific directory path (that is, whose names do not begin with a /). The actual search path depends upon the form of the `#include` directive used for the file
AIX	-qidirfirst		Specifies the search order for files included with the `#include file_name` directive.
AIX	-qinoidirfirst	Default.	Specifies the search order for files included with the `#include file_name` directive.
Tru64	-nocurrent_include		Changes the behavior of the `#include` *`filename`* directive to not search the source file's directory for file name. This option causes the `#include` *`filename`* directives to behave like `#include <filename>` directives.
AIX	-qignprag	all, disjoint, isolated, ibm, or omp.	Instructs the compiler to ignore certain pragmas. Uses : (colon) as delimiter.
AIX	-M		Creates an output file (.u) that contains targets suitable for inclusion in a description file for the AIX `make` command.
Tru64	-M		Outputs a set of make dependency rules to standard output for each source file on the command line (and suppresses compilation). The make dependencies include all of the header files upon which each source file depends. The make targets are the object files for those source files. The output lines are indented to show header file nesting.
Tru64	-MD		Requests dependency files from the preprocessor (and linker if it is also run). It does not suppress compilation like the -M option. This option is passed directly to the preprocessor and linker.
AIX	-qmakedep		Creates an output file (.u) that contains targets suitable for inclusion in a description file for the AIX `make` command. (This is the same as the -M option.)

Operating system	Option	Value and default	Description
AIX	-qmacpstr	Default is nomacpstr.	Converts Pascal string literals into null-terminated strings where the first byte contains the length of the string.
AIX	-qnomacpstr		Disables the conversion of Pascal string literals into null-terminated strings where the first byte contains the length of the string.
AIX	-qmbcs	Default is nombcs.	Use the -qmbcs option if your program contains multibyte characters.
AIX	-qnombcs		Use the -qnombcs option if your program contains multibyte characters.
AIX	-P		Preprocesses the C source files named in the compiler invocation and creates an output preprocessed source file for each input source file.
Tru64	-P		Runs only the C preprocessor and puts the result for each .c or .s source file in a corresponding .i file. The .i file has no #line_number preprocessor directives in it.
AIX	-qpascal		Ignores the word pascal in type specifiers and function declarations.
AIX	-qnopascal		Turns off Ignorance of the word pascal in type specifiers and function declarations.
AIX	-qstdinc	Default is stdinc.	Specifies which files are included with `#include <file_name>` and `#include file_name` directives. If a full (absolute) path name is specified, this option has no effect on that path name. It will still have an effect on all relative path names.
AIX	-qnostdinc	Default is stdinc.	Specifies which files are included with `#include <file_name>` and `#include file_name` directives.
AIX	-qsyntaxonly		Causes the compiler to perform syntax checking without generating an object file.
AIX	-t	c, b, p, a, I, L, l, or m.	Adds the prefix specified by the -B option to designated programs.

Operating system	Option	Value and default	Description
AIX	-U	Name.	Undefines a specified identifier defined by the compiler or by the -D option.
AIX	-qusepcomp	Default is nousecomp.	Uses precompiled header files for any files that have not changed since the precompiled header was created.
AIX	-qnousepcomp		Do not use precompiled header files for any files that have not changed since the precompiled header was created.
AIX	-W	See manual.	Passes the listed words to a designated compiler program.
HP-UX	-C		Prevents the preprocessor from stripping comments.
HP-UX	-Dname		Defines the preprocessor variable name with a value of 1.
HP-UX	-Dname=def		Defines the preprocessor variable name with a value of def.
Tru64	-D	name=def.	Defines the name as if with a `#define` statement. If no definition is given, the name is defined as 1.
HP-UX	-E		Performs preprocessing only with output to stdout.
HP-UX	-s		Strips the symbol table from the executable file.
HP-UX	-U		Undefines name in the preprocessor.
HP-UX	-P		Performs preprocessing only with output to the corresponding .i file.
HP-UX	-z		Disallows run-time dereferencing of null pointers.
HP-UX	-Z		Allows dereferencing of null pointers at run-time.
HP-UX	-t	Program or name.	Substitutes or inserts subprocess program with name.
HP-UX	-W	Program and argstring.	Passes the arguments argstring to the subprocess program.
Solaris	-A	name(token).	Associates name as a predicate with the specified tokens as if by a `#assert` preprocessing directive.

Operating system	Option	Value and default	Description
Solaris	-C		Prevents the preprocessor from removing comments, except those on the preprocessing directive lines.
Solaris	-D	name=token.	Associates name with the specified tokens, as if by a `#define` preprocessing directive. if no token is specified, the value 1 will be used.
Solaris	-E		Runs the source file through the preprocessor only and sends the output to stdout.
Solaris	-fd		Reports K&R-style function definitions and declarations.
Solaris	-H		Prints to standard error, one per line, the path name of each file included during the current compilation.
Solaris	-I	Dir.	Adds dir to the list that is searched for #include files with relative file names.
Solaris	-P		Runs the source file through the C preprocessor only.
Solaris	-U	Name.	Removes any initial definition of the preprocessor symbol name.
Tru64	-U	Name.	Removes any macro definition of name at the start of the compilation. name could have been defined with a -D option or predefined by the compiler. If no name is specified or if name is not defined, the -U option is ignored.
Solaris	-xCC		Accepts the C++-style comments.
Solaris	-xM		Runs only the preprocessor on the named C programs, requesting that it generate makefile dependencies and send the result to the standard output.
Solaris	-xM1		Collects dependencies like -xM, but excludes /usr/include files.
Solaris	-xP		Prints prototypes for all K&R C functions defined in this module.
Solaris	-xpg		Prepares the object code to collect data for profiling with gprof(1).

Operating system	Option	Value and default	Description
Solaris	-xsb		Generates extra symbol table information for the source browser.
Solaris	-xsbfast		Creates the database for the source browser.
Tru64	-FI	Filename.	Specifies a file that is to be included before the first line in a source file is actually compiled.
Tru64	-machine_code		Includes the generated machine code in the listing file. By default, machine code is not listed. To produce the listing file, you must also specify -source_listing.
Tru64	-cpp	This is the default.	Calls the C macro preprocessor on C and assembly source files before compiling.
Tru64	-nocpp		Does not call the C macro preprocessor on C and assembly source files before compiling.
Tru64	-oldcomment		Directs the preprocessor to delete comments (replacing them with nothing at all). This allows traditional token concatenation.

C.8 Compiled code compiler options

Table 134 shows the options that deals with the compiled code.

Table 134. Compiled code options

Operating system	Option	Value and default	Description
AIX	-qbitfields	unsigned or signed. Default is unsigned.	Specifies if bit fields are signed or unsigned.
AIX	-qdollar		Allows the $ symbol to be used in the names of identifiers.
AIX	-qnodollar		Does not allow the $ symbol to be used in the names of identifiers.
AIX	-qupconv	noupconv.	Preserves the unsigned specification when performing integral promotions.
AIX	-qnoupconv		Suppresses the prevention of the unsigned specification when performing integral promotions.

Operating system	Option	Value and default	Description
AIX	-c		Instructs the compiler to pass source files to the compiler only.
Tru64	-c		Suppresses the loading phase of the compilation and forces the creation of an object file.
AIX	-qchars	signed or unsigned. Default is unsigned.	Instructs the compiler to treat all variables of type char as either signed or unsigned.
AIX	-qprocimported	““ or funcname. See the manual for the default.	Marks funcname as a imported function.
AIX	-qproclocal	““ or funcname. See the manual for the default.	Marks funcname as a local function.
AIX	-qprocunknown	““ or funcname. See the manual for the default.	Marks funcname as a unknown function.
AIX	-r		Produces a relocatable object.
AIX	-S		Generates an assembly language file (.s) for each source file.
AIX	-qstatsym	Default is nostatsym.	Adds user-defined, non-external names that have a persistent storage class to the name list.
AiX	-qnostatsym		Disables adding of user-defined, non-external names that have a persistent storage class to the name list.
AIX	-qtbtable	none, full, or small. Default is full.	Sets traceback table characteristics.
HP-UX	-o	outfile.	Places object modules in outfile file.
HP-UX	-S		Generates an assembly language source file.
HP-UX	+sb		Make bit-fields signed by default in both 32- and 64-bit modes.

Operating system	Option	Value and default	Description
HP-UX	+uc		Makes unqualified char data types unsigned.
Solaris	-c		Directs the compiler to suppress linking with ld(1) and to produce a .o file for each source file.
AIX	-qdataimported	names. Default is datalocal.	Marks data as imported.
AIX	-qdatalocal		Marks data as local.
AIX	-o	filename.	Specifies a name or directory for the output executable file(s) created either by the compiler or the linkage editor.
Tru64	-o	filename.	Names the final output file output.
HP-UX	-c		Compiles only; does not link.
Solaris	-o	filename.	Names the output file
Solaris	-S		Directs the compiler to produce an assembly source file but not to assemble the program.
Tru64	-S		Compiles the specified source files and generates symbolic assembly language output in corresponding files suffixed with .s.
Tru64	-noobject		Suppresses creation of an object file. By default, an object module file is created with the same name as that of the first source file of a compilation unit and with the .o file extension.
Tru64	-Q		Directs the preprocessor to use single quotes in __FILE__ expansions instead of double quotes.
Tru64	-signed	This is the default.	Causes all char declarations to have the same representation and range of values as signed char declarations. This is used to override a previous -unsigned option. This is the default.
Tru64	-unsigned		Causes all char declarations to have the same representation and range of values as unsigned char declarations.
Tru64	-volatile		Causes all variables to be treated as volatile.

C.9 Compilation mode compiler options

Table 135 shows the options that deals with the compilation. such as messages, temporary files, and so on.

Table 135. Compiler mode options

Operating system	Option	Value and default	Description
AIX	-#		Traces the compilation without doing anything.
AIX	-qinfo	See manual.	Produces informational messages.
AIX	-F	config_file, stanza, or config_file: stanza.	Names an alternative configuration file for `xlc`.
AIX	-qnoinfo		Does not produces informational messages.
AIX	-qproto	noproto.	Assumes all functions are prototyped.
AIX	-qnoproto		Does not assume that all functions are prototyped.
AIX	-qtabsize=	Default is 8.	Changes the length of tabs, as perceived by the compiler.
HP-UX	+help		Launches a Web browser displaying an HTML version of the HP C/HP-UX online help.
HP-UX	-V		Causes subprocesses to print version information to stderr.
Solaris	-#		Turns on verbose mode, showing each component as it is invoked.
Solaris	-###		Shows each component as it would be invoked, but does not actually execute it.
Solaris	-keeptmp		Retains temporary files created during compilation instead of automatically deleting them.
Solaris	-V		Directs `cc` to print the name and version ID of each component as the compiler executes.
Solaris	-xhelp=f	flags, readme, or errors.	Displays online help information.
Solaris	-xtemp=	directory.	Sets the directory for temporary files used by `cc` to dir.-xtemp has precedence over the TMPDIR environment variable.

Operating system	Option	Value and default	Description
Solaris	-xtime		Reports the time and resources used by each compilation component.
Tru64	-accept	[no]vaxc_keywords or [no]restrict_keyword.	Causes the compiler to recognize additional keywords.
Tru64	-edit	0-9.	When syntactic or semantic errors are detected by the compiler's front end, invokes the editor defined by the environment variable EDITOR (or vi if EDITOR is undefined). Two files are opened for editing: the error message file, which indicates the location of the error, and the source file. When you exit from the editor, the compilation is restarted. The n argument specifies the number of times a compilation can be interrupted in this way
Tru64	-error_limit	number. Default is 30.	Sets a limit on the number of error-level diagnostics that the compiler will emit.
Tru64	-noerror_limit		Unsets a limit on the number of error-level diagnostics that the compiler will emit.
Tru64	-nestlevel	Number. Default is 50.	Sets the nesting-level limit for include files.
Tru64	-V		Prints the version of the compiler driver.
Tru64	-proto	i or s.	Extracts prototype declarations for function definitions and puts them in a .H suffixed file. The suboption i includes identifiers in the prototype, and the suboption s generates prototypes for static functions as well.
Tru64	-H		Halts compiling after the pass specified by the character c, producing an intermediate file for the next pass. The c character can be one of the following: [fablL] (see the -t option for an explanation). It selects the compiler pass in the same way as the -t option. If this option is used, the symbol table file produced and used by the passes is given the name of the last component of the source file with the suffix changed to .T, and the file is always retained after the compilation is halted.

Operating system	Option	Value and default	Description
Tru64	-W		Passes the argument, or arguments (argi), to the compiler pass, or passes (c[c...]). Each c character can be one of the following: [ablLzpfy] (see the -t option for an explanation). The c selects the compiler pass in the same way as the -t option.
Tru64	-t		The -t, -h, and -B options are used together to specify a location and/or name for one or more compiler passes, tools, libraries, or include files, other than their normal locations or names. The -t option specifies which compiler passes (or components) the -h and -B options that follow apply to.
Tru64	-h path		Specifies the directory where the tool (or other component) specified with -t is located. If -h is omitted, the tool is assumed to be in the usual location (for example, /usr/lib/complrs/cc). If path is omitted, the tool is assumed to be in the root directory (/).
Tru64	-B		Specifies a suffix to add to the normal names of any components specified with the -t option. If string is omitted, the usual component names are used.
Tru64	-K		Directs the compiler to give recognizable names to intermediate files and retain them for debugging purposes. Each file is given the name of the last component of the source file, replacing its suffix with the conventional suffix for the type of file (for example, .B suffix for binary ucode produced by the front end).

C.10 Diagnostics compiler options

Compiler options that deal with diagnostics are shown in Table 136.

Table 136. Compiler diagnostics options

Operating system	Option	Value and default	Description
AIX	-qflag	i, w, e, s, or u. Default is flag=i:i.	Specifies the minimum severity level of diagnostic messages to be reported in source listings (first specifier) and on screen (second specifier).
AIX	-qmaxerr	Default is nomaxerr.	Instructs the compiler to halt compilation when a specified number of errors of specified or greater severity is reached.
AIX	-qnomaxerr		Allow as many errors as needed.

Operating system	Option	Value and default	Description
AIX	-qnoprint		Suppresses listings.
AIX	-qshowinc	Default is noshowinc.	If used with the -qsource option, all the include files are included in the source listing.
AIX	-qnoshowinc		Disables showing of include files in source listing.
AIX	-qsource	Default is nosource.	Produces a compiler listing and includes source code.
AIX	-qnosource		Disables production of compiler listing and includes source code.
AIX	-qsrcmsg	nosrcmsg.	Adds the corresponding source code lines to the diagnostic messages in the stderr file.
AIX	-qnosrcmsg		Disables the addition of the corresponding source code lines to the diagnostic messages in the stderr file.
AIX	-qsupress	msg_num. Default is nosupress.	Lets you specify warning or information messages to be suppressed in the compiler listing.
AIX	-qnosupress		Disables suppressing of warning or information messages to be suppressed in the compiler listing.
AIX	-qwarn64		Enables warning of possible long to integer data truncations.
AIX	-qnowarn64	default.	Disables warning of possible long to integer data truncations.
AIX	-v		Instructs the compiler to report information on the progress of the compilation.
Tru64	-v		Prints the compilation phases as they execute with their arguments and their input and output files. Prints resource usage in the C-shell time format. Prints the macros defined at the start of the compilation.
AIX	-w		Requests that warning messages be suppressed.
AIX	-qxcall	Default is noxcall.	Generates code to static routines within a compilation unit as if they were external calls.
AIX	-qnoxcall		Does not generate code to static routines within a compilation unit as if they were external calls. This generates faster code than -qxcall

Operating system	Option	Value and default	Description
AIX	-qspnans	nospnans.	Generates extra instructions to detect signalling NaN on conversion from single precision to double precision.
AIX	-qspnans		Disables the generation of extra instructions to detect signalling NaN on conversion from single precision to double precision.
AIX	-qsyntaxonly		Causes the compiler to perform syntax checking without generating an object file.
HP-UX	+m		Prints identifier maps in the source code listing.
HP-UX	+L		Enables any `#pragma` listing directives and the listing facility.
HP-UX	+o		Prints hexadecimal code offsets in the source code listing.
HP-UX	-v		Enables verbose mode.
HP-UX	+M2		Provides migration warnings for transitioning code from the ILP32 to the LP64 data model.
HP-UX	+M		Provides ANSI migration warnings that explain the differences between code compiled with -Ac and -Aa.
HP-UX	+M1		Provides platform migration warnings for features that may not be supported in future releases.
HP-UX	-w		Suppresses warning messages.
HP-UX	+Wen	1[,2,...N].	Changes the specified warnings to errors.
HP-UX	+wn		Specifies the level of the warning messages where n is 1 - 3.
HP-UX	+Wn	1[,2,...N].	Suppresses the specified warnings.
HP-UX	+Wwn	1[,2,...N].	Enables the specified warnings, assuming all other warnings are suppressed with -w or +w3.
Solaris	-erroff=t	tag, no%tag, %all, or %none. Default is none.	Suppresses compiler warning messages.

Operating system	Option	Value and default	Description
Solaris	-errtags=	yes or no. Default is yes if -errtags is specified	Displays the message tag for each warning message.
Solaris	-errwarn=t	tag, no%tag, %all, or %none. Default is none.	If the indicated warning message is issued, `cc` exits with a failure status.
Solaris	-v		Directs the compiler to perform stricter semantic checks and to enable other `lint`-like checks.
Solaris	-w		Suppresses compiler warning messages.
Solaris	-xe		Performs only syntax and semantic checking on the source file, but does not produce any object or executable code.
Solaris	-xtransition		Issues warnings for the differences between K&R C and Sun ANSI/ISO C.
Solaris	-xvpara		Warns about loops that have `#pragma` MP directives specified, but may not be properly specified for parallelization.
Tru64	-show	See manual.	Specifies one or more items to be included in the listing file. When specifying multiple keywords, separates them by commas and no intervening blanks.
Tru64	-source_listing		Produces a source program listing file with the same name as the source file and with a .lis file extension. You must specify this qualifier to get a listing. The default is to not produce a listing file.
Tru64	-check		Performs compile-time code checking. With this option, the compiler checks for code that exhibits nonportable behavior, represents a possible unintended code sequence, or possibly affects operation of the program because of a quiet change in the ANSI C standard.
Tru64	-msg_dump		Causes the compiler to dump, to stdout, all messages enabled by any given `cc` command line. The compiler then exits, without doing a compilation.

Operating system	Option	Value and default	Description
Tru64	-msg_enable	level0-level6, 64bit, alignment, c_to_cxx, check, defunct, noansi, obsolescent, overflow, performance, portable, preprocessor, questcode, returnchecks, or unused.	Enables a specific message or group of messages.
Tru64	-msg_disable	Same as above.	Disables a specific message or group of messages. (Note that messages with error or fatal severity cannot be disabled; only warning and informational messages can be disabled.)
Tru64	-msg_always	Same as above.	Always emits the messages identified by the msg_list argument.
Tru64	-msg_once	Same as above.	Emits the identified messages only once.
Tru64	-msg_fatal	Same as above.	Changes the identified messages to be fatal, compilation-ending errors.
Tru64	-msg_warn	Same as above.	Changes the identified messages to be warnings. (Note that error- or fatal-severity messages cannot be changed to warning-severity messages.)
Tru64	-msg_inform	Same as above.	Changes the identified messages to be informational messages. (Note that error- or fatal-severity messages cannot be changed to informational-severity messages.)
Tru64	-msg_error	Same as above.	Changes the identified messages to be error messages. (Note that fatal-severity messages cannot be changed to error-severity messages.)
Tru64	-portable		Directs the compiler to issue diagnostics for certain constructs that may not be portable to other compilers or platforms. -portable is equivalent to -msg_enable portable.

Operating system	Option	Value and default	Description
Tru64	-SD	directory.	Suppresses certain warning- and informational-level diagnostic messages that are inappropriate for system header files. The suppressed messages relate to non-portable constructs in header files whose path names are prefixed by string directory.
Tru64	-verbose		Produces longer error and warning messages. Messages in this form may give the user more hints about why the compilation failed.
Tru64	-w	0, 1, 2, or 3.	Controls the display of messages as well as the actions that occur as a result of the messages.
Tru64	-warnprotos		Causes the compiler to produce warning messages when a function is called that is not declared with a full prototype. This checking is more strict than required by ANSI C.
Tru64	-varargs		Prints warnings for all lines that may require the <varargs.h> macros.

C.11 Debugging compiler options

Options that deals with debugging are shown in Table 137.

Table 137. Debugging options

Operating system	Option	Value and default	Description
AIX	-qcheck	all, (NO)NULLptr, (no)bounds, (NO)DIVzero. Default is -qnocheck.	Generates code which performs certain types of run-time checking. Use : as delimiter.
AIX	-qnocheck		Disables generation of code which performs certain types of run-time checking.
AIX	-qdpcl	nodpcl.	Generates symbols that tools based on the Dynamic Probe Class Library (DPCL) can use to see the structure of an executable file.
AIX	-qnodpcl		Disables generation of symbols that tools based on the Dynamic Probe Class Library (DPCL) can use to see the structure of an executable file.

Operating system	Option	Value and default	Description
AIX	-g		Generates debugging information used by the debugger.
AIX	-qdbxextra		Specifies that all typedef declarations, struct, union, and enum type definitions are included for debugger processing.
AIX	-qnodbxextra	nodbxextra.	Specifies that no typedef declarations, struct, union, and enum type definitions are included for debugger processing.
AIX	-qextchk	noextchk.	Generates bind-time type checking information and checks for compile-time consistency.
AIX	-qnoextchk		Disables generation of bind-time type checking information and checks for compile-time consistency.
AIX	-qhalt	i, w, e, s, or u. Default is s.	Instructs the compiler to stop after the compilation phase when it encounters errors of specified severity or greater.
AIX	-qheapdebug	noheapdebug.	Enables debug versions of memory management functions.
AIX	-qnoheapdebug		Enables debug versions of memory management functions.
AIX	-qlinedebug	Default is nolinedebug.	Generates abbreviated line number and source file name information for the debugger.
AIX	-qnolinedebug		Suppresses generation of abbreviated line number and source file name information for the debugger.
AIX	-qlist	nolist.	Produces a compiler listing that includes an object listing.
AIX	-qnolist		Produces a compiler listing that includes an object listing.
AIX	-qlistopt	Default is nolistopt.	Suppresses a compiler listing that displays all options in effect.
AIX	-qnolistopt		Suppresses a compiler listing that displays all options in effect.
AIX	-qphsinfo	Default is nophsinfo.	Reports the time taken in each compilation phase.

Operating system	Option	Value and default	Description
AIX	-qnophsinfo		Does not report the time taken in each compilation phase.
AIX	-qfullpath	Default is notullpath.	Specifies what path information is stored for files when you use the -g option.
AIX	-qnofullpath		Disables the use of full path information when you use the -g option.
AIX	-qsymtab=	unref or static.	If specified, includes all typedef, struct, union, and enum type definitions. If static is specified, also includes user-defined, nonexternal names that have a persistent storage class.
AIX	-qxref	Default is noxref.	Produces a compiler that includes a cross-reference of all identifiers.
AIX	-qnoxref		Do not produce a compiler listing that includes a cross-reference listing of all identifiers.
HP-UX	-g		Inserts information for the symbolic debugger in the object file.
HP-UX	+objdebug		When used with -g, +objdebug leaves debug information in the object files instead of copying it to the executable file. The object files must be accessible to the HP WDB debugger when debugging. This option is not supported by the HP DDE debugger.
HP-UX	+noobjdebug		Disables +objdebug.
Solaris	-g		Produces additional symbol table information for the debugger.
Tru64	-g	0, 1, 2, or 3. Default is 2.	Produces symbol table information for debugging.
Solaris	-s		Removes all symbolic debugging information from the output object file.
Solaris	-xs		Disables auto-read for `dbx`.
Tru64	-check_bounds		Generates run-time code to check the values of array subscripts (and equivalent pointer arithmetic involving pointers produced by converting an array name to a pointer) to verify that the resulting address lies within the range for which the C standard requires well-defined behavior.

Operating system	Option	Value and default	Description
Tru64	-nocheck_bounds		Disables generation of run-time bound checking.

C.12 Linking and libraries compiler options

Compiler options that deals with linking and other library related options are shown in Table 138 to Table 141.

C.12.1 Placing string literals and constants

Table 138. String literal options

Operating system	Option	Value and default	Description
AIX	-qro	Default for `xlc` and `c89`	Specifies the storage type for string literals. Strings are placed in ro storage.
AIX	-qnoro	Default for `cc`	Specifies the storage type for string literals. Strings are placed in rw storage.
AIX	-qroconst	Default for `xlc` and `c89`	Specifies the storage type for constant values. Constant values are placed in ro storage.
AIX	-qnoroconst	Default for `cc`	Specifies the storage type for constant values. Constant values are placed in rw storage.
HP-UX	+ESconstlit		Introduces new default behavior. HP C now stores constant-qualified (const) objects and literals in read-only memory.
HP-UX	+ESlit		Places string literals and constants into read-only data storage.
HP-UX	+ESnolit		Disables +ESconstlit, causing HP C to no longer store literals in read-only memory.
Solaris	-xstrconst		Inserts string literals into the read-only data section of the text segment instead of the default data segment.
Tru64	-readonly_strings		Allows the compiler to assume that string literals are read-only. This may improve application performance. This option overrides -writable_strings, which is the default.
Tru64	-writable_strings		Causes all string literals to be writable. This is the default. This option overrides -readonly_strings.

C.12.2 Static and dynamic linking and libraries

Table 139. Linking options

Operating system	Option	Value and default	Description
AIX	-brtl		Tells the linkage editor to accept both .so and .a library file types.
AIX	-bstatic		Specifies that static types of library files are searched by the linkage editor.
AIX	-bdynamic	Dynamic is default.	Specifies that dynamic types of library files are searched by the linkage editor.
AIX	-G		Linkage editor (ld command) option only. Used to generate a dynamic library file.
AIX	-qmkshrobj=		Creates a shared object from generated object files. This option, together with the related options described below, should be used instead of the `makeC++SharedLib` command. The advantage to using this option is that the compiler will automatically include and compile the template instantiations in the tempinc directory.
HP-UX	-n		Generates shareable code.
HP-UX	-N		Generates unshareable code.
HP-UX	-noshared		Creates statically-bound executables.
Tru64	-non_shared		Directs the linker to produce a static executable. The output object created by the linker will not use any shared objects during execution.
HP-UX	+z		Generates shared library object code (same as +Z in 64-bit mode).
HP-UX	+Z		Generates shared library object code with a large data linkage table (long-form PIC).
HP-UX	-dynamic		Enables linking of PIC objects.

Operating system	Option	Value and default	Description
HP-UX	-q		Marks the executable as demand loadable.
HP-UX	-Q		Marks the executable as not being demand loadable.
Solaris	-B	static or dynamic.	Specifies whether bindings of libraries for linking are static or dynamic.
Solaris	-d	y or n.	Specifies dynamic or static linking in the link editor.
Solaris	-G		Passes the option to the link editor to produce a shared object rather than a dynamically linked executable.
Solaris	-h	libname.	Assigns a name to a shared dynamic library as a way to have different versions of a library.
Tru64	-call_shared		Produces a dynamic executable file that uses shareable objects during run time. This is the default. The loader uses shareable objects to resolve undefined symbols.
Tru64	-expect_unresolved	pattern.	Causes any unresolved symbols matching pattern to be ignored. Such symbols are not displayed and are not treated as errors or warnings. You can enter this option multiple times on a command line.
Tru64	-exact_version		Used in conjunction with -call_shared to request strict dependency testing for the executable file produced. Executable files built in this manner can be executed only if the shared libraries that they use were not modified after the executable was built.

Operating system	Option	Value and default	Description
Tru64	-shared		Produces dynamic shareable objects. The loader will produce a shareable object that other dynamic executables can use at run time.
Tru64	-check_registry	location_file.	Checks the location of this shared object's segments and make sure they stay out of the way of other object's segments in the location_file. Multiple instances of this option are allowed (Needs -shared).
Tru64	-rpath	path.	Creates an rpath record containing the specified path string. The path string is a colon-separated list of directories that is meaningful only when creating an executable with shared linkage (Needs -shared).
Tru64	-set_version	version string.	Establishes the version identifier (or identifiers) associated with a shared library. The string version-string is either a single version identifier or a colon-separated list of version identifiers (Needs -shared).
Tru64	-soname	shared_object_name.	Sets DT_SONAME for a shared object. The name can be a single component name (for example, libc.a), a full path name (starting with a slash), or a relative path name (containing a slash) (Needs -shared).
Tru64	-update_registry	location_file.	Registers the location of this shared object's segments and makes sure they stay out of the way of others in the location_file. Location_file is updated if it is writable (Needs -shared).

C.12.3 Directories

Table 140. Directory search options

Operating system	Option	Value and default	Description
AIX	-l	key	Searches a specified library for linking.
HP-UX	-I	dir	Inserts dir in the include file search path.
Solaris	-l	library	Links with object library libname.so or libname.a.
AIX	-L	directory	Searches the specified directory for library files specified by the -l option.
HP-UX	-L	directory	Links the libraries in directory before the libraries in the default search path.
Solaris	-L	directory	Adds directories to the list that the linker searches for libraries.
Solaris	-i		Passes the option to the linker to ignore any LD_LIBRARY_PATH setting.
Solaris	-R	dir,dir...	Passes a colon-separated list of directories used to specify library search directories to the run-time linker.

C.12.4 Other linker options

Table 141. Other linker options

Operating system	Option	Value and default	Description
AIX	-f	filename.	Linkage editor (`ld` command) option only. Passes to the linkage editor the file name of a file containing a list of input files to be processed.
AIX	-qinlglue	noinlglue.	Generates fast external linkage by inlining the pointer glue code necessary to make a call to an external function or a call through a function pointer.
AIX	-qnoinlglue		Does note generate ‘linker glue.’
HP-UX	-lx		Links with the /lib/libx.a and /usr/lib/libx.a libraries.
HP-UX	-Wd		Omits HP provided prefix files required by the linker.
HP-UX	-a		Omits HP provided prefix files required by the linker.
HP-UX	+a		Omits HP provided prefix files required by the linker.

Operating system	Option	Value and default	Description
Solaris	-mc		Removes duplicate strings from the .comment section of the object file.
Solaris	-mr		Removes all strings from the .comment section.
Solaris	-mr,	string.	Removes all strings from the .comment section and inserts string in that section of the object file.
Solaris	-Q	y or n. Default is y.	Emits or does not emit identification information to the output file.
Solaris	-xMerge		Merges data segments into text segments.
Solaris	-xcode=v	abs32, abs44, abs64, pic13, or pic32. Default depends on processor.	Specify code address space (SPARC ONLY).
Solaris	-xildoff		Turns off the incremental linker and forces the use of `ld`.
Solaris	-xildon		Turns on the incremental linker and forces the use of `ild` in incremental mode.
Solaris	-xnolib		Does not link any libraries by default
Tru64	-fini	symbol.	Makes the procedure represented by the symbol into a termination routine. A termination routine is a routine that is called without an argument when either the file that contains the routine is unloaded or the program that contains the routine exits.
Tru64	-init	symbol.	Makes the procedure represented by the symbol into an initialization routine. An initialization routine is a routine that is called without an argument when either the file that contains the routine is loaded or the program that contains the routine is started.
Tru64	-input_to_ld	filename.	Directs the linker to read the contents of file filename as if the contents had been supplied on the `ld` command line.

Operating system	Option	Value and default	Description
Tru64	-noarchive		Prevents the linker from using archive libraries to resolve symbols. This option is used in conjunction with -call_shared. The -noarchive option is position sensitive; it affects only those options and variables that follow it on the command line. This option can also be used more than once on the command line.

C.13 Target platform compiler options

Compiler options that deal with the target architecture are shown in Table 142.

Table 142. Target environment options

Operating system	Option	Value and default	Description
AIX	-q32	default.	Selects 32-bit compiler mode.
HP-UX	+DD32		Generates 32-bit code for PA1.1 architecture.
AIX	-q64		Selects 64-bit compiler mode.
HP-UX	+DD64		Generates 64-bit code for PA2.0 architecture.
AIX	-qarch=	auto, com or, pwr, pwr2, pwrx, ppc, ppcgr, or noauto. Default is com.	Specifies the architecture on which the executable program will be run.
HP-UX	+DA	model.	Generates object code for a specific version of the PA-RISC architecture.
HP-UX	+DC	apptype.	Generates code for portable or embedded applications.

Operating system	Option	Value and default	Description
Solaris	-xarch=	generic, native, v7, v8a, v8, v8plus, v8plusa, v8plusb, v9, v9a, v9b, 386, pentium, or pentium_pro.	Specifies instruction set architecture.
Solaris	-xcg=	89 or 92.	Macro that specifies values for -xarch, -xchip, and -xcache (SPARC only).
HP-UX	+DO	osname.	Sets the target operating system for the compiler.
Solaris	-xregs=	appl, no%appl, float, or no%float. Default is appl or float.	Specifies the usage of registers for the generated code (SPARC only).
Tru64	-arch	generic, host, ev4, ev5, ev56, ev56, ev67, or pca56.	Specifies which version of the Alpha architecture to generate instructions for. All Alpha processors implement a core set of instructions and, in some cases, extensions.
Tru64	-taso		Directs the linker to load the executable file in the lower 31-bit addressable virtual address range. The -T and -D options to the `ld` command can also be used, respectively, to ensure that the text and data segments are loaded into low memory.
Tru64	-xtaso		Causes the compiler to respect `#pragma pointer_size` directives, which control the size of pointers. These directives are ignored otherwise. Also causes the -taso option to be passed to the linker (if linking).
Tru64	-xtaso_short		Same as the -xtaso option, except -xtaso_short also directs the compiler to allocate 32-bit pointers by default. You can still use 64-bit pointers, but only by the use of pointer_size pragmas.

Operating system	Option	Value and default	Description
Tru64	-framepointer		Makes all procedures in the source file use $fp (register 15) as the frame pointer.

C.14 GCC options specific for AIX 5L

This section details the platform specific options of GCC when running on RS/6000 and pSeries machines.

C.14.1 AIX options

The options that the GNU GCC compiler supports for AIX are shown in Table 143.

Table 143. AIX options for GNU GCC

Option	Description
-gxcoff	Produces debugging information in XCOFF format (if that is supported). This is the format used by the DBX debugger on IBM RS/6000 systems.
-gxcoff+	Produces debugging information in XCOFF format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program, and may cause assemblers other than the GNU assembler (GAS) to fail with an error.
-mnohc-struct-return	Returns some structures of more than one word in registers, when convenient. This is the default. For compatibility with the IBM-supplied compilers, use the option -fpcc-struct-return or the option -mhc-struct-return.
-mthreads	Support AIX threads. Links an application written to use pthreads with special libraries and startup code to enable the application to run.

C.14.2 Power and PowerPC options

The options that the GNU GCC compiler supports for the POWER and PowerPC processors on the RS/6000 and pSeries machines are shown in Table 144.

Table 144. RS/6000 and pSeries specific options for GNU GCC

Option	Description
-fpic	Generates position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of GCC; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-specific maximum size, you get an error message from the linker indicating that -fpic does not work; in that case, recompile with -fPIC instead. (These maximums are 16 KB on the m88k, 8 KB on the SPARC, and 32 KB on the m68k and RS/6000. The 386 has no such limit.) Position-independent code requires special support, and therefore works only on certain machines. For the 386, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM RS/6000 is always position-independent.

Option	Description
-mpower, -mno-power, -mpower2, -mno-power2, -mpowerpc, -mno-powerpc, -mpowerpc-gpopt, -mno-powerpc-gpopt, -mpowerpc-gfxopt, -mno-powerpc-gfxopt, -mpowerpc64, or -mno-powerpc64	GCC supports two related instruction set architectures for the RS/6000 and PowerPC. The POWER instruction set are those instructions supported by the rios chip set used in the original RS/6000 systems. The PowerPC instruction set is the architecture of the Motorola MPC5xx, MPC6xx, and MPC8xx microprocessors, and the IBM 4xx microprocessors. Neither architecture is a subset of the other. However, there is a large common subset of instructions supported by both. An MQ register is included in processors supporting the POWER architecture. You use these options to specify which instructions are available on the processor you are using. The default value of these options is determined when configuring GCC. Specifying -mcpu=cpu_type overrides the specification of these options. We recommend you use the -mcpu=cpu_type option rather than the options listed above. The -mpower option allows GCC to generate instructions that are found only in the POWER architecture and to use the MQ register. Specifying -mpower2 implies -power, and also allows GCC to generate instructions that are present in the POWER2 architecture but not the original POWER architecture. The -mpowerpc option allows GCC to generate instructions that are found only in the 32-bit subset of the PowerPC architecture. Specifying -mpowerpc-gpopt implies -mpowerpc and also allows GCC to use the optional PowerPC architecture instructions in the General Purpose group, including floating-point square root. Specifying -mpowerpc-gfxopt implies -mpowerpc and also allows GCC to use the optional PowerPC architecture instructions in the Graphics group, including floating-point select. The -mpowerpc64 option allows GCC to generate the additional 64-bit instructions that are found in the full PowerPC64 architecture and to treat GPRs as 64-bit, doubleword quantities. GCC defaults to -mno-powerpc64. If you specify both -mno-power and -mno-powerpc, GCC will use only the instructions in the common subset of both architectures plus some special AIX common-mode calls, and will not use the MQ register. Specifying both -mpower and -mpowerpc permits GCC to use any instruction from either architecture and to allow use of the MQ register; specify this for the Motorola MPC601.
-mnew-mnemonics or -mold-mnemonics	Selects which mnemonics to use in the generated assembler code. -mnew-mnemonics requests output that uses the assembler mnemonics defined for the PowerPC architecture, while -mold-mnemonics requests the assembler mnemonics defined for the POWER architecture. Instructions defined in only one architecture have only one mnemonic; GCC uses that mnemonic irrespective of which of these options is specified. GCC defaults to the mnemonics appropriate for the architecture in use. Specifying -mcpu=cpu_type sometimes overrides the value of these option. Unless you are building a cross-compiler, you should normally not specify either -mnew-mnemonics or -mold-mnemonics' but should instead accept the default.

Option	Description
-mcpu=cpu_type	Sets architecture type, register usage, choice of mnemonics, and instruction scheduling parameters for machine type cpu_type. Supported values for cpu_type are rios, rios1, rsc, rios2, rs64a, 601, 602, 603, 603e, 604, 604e, 620, 630, 740, 750, power, power2, powerpc, 403, 505, 801, 821, 823, and 860 and common. -mcpu=power, -mcpu=power2, -mcpu=powerpc, and -mcpu=powerpc64 specify generic POWER, POWER2, pure 32-bit PowerPC (that is, not MPC601), and 64-bit PowerPC architecture machine types, with an appropriate, generic processor model assumed for scheduling purposes. Specifying any of the following options: -mcpu=rios1, -mcpu=rios2, -mcpu=rsc, -mcpu=power, or -mcpu=power2 enables the -mpower option and disables the -mpowerpc option; -mcpu=601 enables both the -mpower and -mpowerpc options. All of -mcpu=rs64a, -mcpu=602, -mcpu=603, -mcpu=603e, -mcpu=604, -mcpu=620, -mcpu=630, -mcpu=740, and -mcpu=750 enable the -mpowerpc option and disable the -mpower option. Similarly, all of -mcpu=403, -mcpu=505, -mcpu=821, -mcpu=860 and -mcpu=powerpc enable the -mpowerpc option and disable the -mpower option. -mcpu=common disables both the -mpower and -mpowerpc options. AIX Version 4 or greater selects -mcpu=common by default, so that code will operate on all members of the RS/6000 POWER and PowerPC families. In that case, GCC will use only the instructions in the common subset of both architectures plus some special AIX common-mode calls, and will not use the MQ register. GCC assumes a generic processor model for scheduling purposes. Specifying any of the options -mcpu=rios1, -mcpu=rios2, -mcpu=rsc, -mcpu=power, or -mcpu=power2 also disables the new-mnemonics option. Specifying -mcpu=601, -mcpu=602, -mcpu=603, -mcpu=603e, -mcpu=604, -mcpu=620, -mcpu=630, -mcpu=403, -mcpu=505, -mcpu=821, -mcpu=860 or -mcpu=powerpc also enables the new-mnemonics option. Specifying -mcpu=403, -mcpu=821, or -mcpu=860 also enables the -msoft-float option.
-mtune=cpu_type	Sets the instruction scheduling parameters for machine type cpu_type, but does not set the architecture type, register usage, choice of mnemonics like -mcpu=cpu_type would. The same values for cpu_type are used for -mtune=cpu_type as for -mcpu=cpu_type. The -mtune=cpu_type option overrides the -mcpu=cpu_type option in terms of instruction scheduling parameters.

Option	Description
-mfull-toc, -mno-fp-in-toc, -mno-sum-in-toc, or -mminimal-toc	Modifies generation of the TOC (Table Of Contents), which is created for every executable file. The `-mfull-toc' option is selected by default. In that case, GCC will allocate at least one TOC entry for each unique non-automatic variable reference in your program. GCC will also place floating-point constants in the TOC. However, only 16,384 entries are available in the TOC. If you receive a linker error message saying that you have overflowed the available TOC space, you can reduce the amount of TOC space used with the -mno-fp-in-toc and -mno-sum-in-toc options. -mno-fp-in-toc prevents GCC from putting floating-point constants in the TOC and -mno-sum-in-toc forces GCC to generate code to calculate the sum of an address and a constant at run time, instead of putting that sum into the TOC. You may specify one or both of these options. Each causes GCC to produce very slightly slower and larger code at the expense of conserving TOC space. If you still run out of space in the TOC even when you specify both of these options, specify -mminimal-toc instead. This option causes GCC to make only one TOC entry for every file. When you specify this option, GCC will produce code that is slower and larger but which uses extremely little TOC space. You may wish to use this option only on files that contain less frequently executed code.
-maix64 or -maix32	Enables 64-bit AIX ABI and calling convention: 64-bit pointers, 64-bit long type, and the infrastructure needed to support them. Specifying -maix64 implies -mpowerpc64 and -mpowerpc, while -maix32 disables the 64-bit ABI and implies -mno-powerpc64. GCC defaults to -maix32.
-mxl-call or -mno-xl-call	On AIX, passes floating-point arguments to prototyped functions beyond the register save area (RSA) on the stack in addition to argument FPRs. The AIX calling convention was extended but not initially documented to handle an obscure K&R C case of calling a function that takes the address of its arguments with fewer arguments than declared. AIX XL compilers access floating point arguments which do not fit in the RSA from the stack when a subroutine is compiled without optimization. Because always storing floating-point arguments on the stack is inefficient and rarely needed, this option is not enabled by default and only is necessary when calling subroutines compiled by AIX XL compilers without optimization.
-mpe	Supports IBM RS/6000 SP Parallel Environment (PE). Links an application written to use message passing with special startup code to enable the application to run. The system must have PE installed in the standard location (/usr/lpp/ppe.poe/), or the specs file must be overridden with the -specs= option to specify the appropriate directory location. The Parallel Environment does not support threads, so the -mpe option and the -mthreads option are incompatible.
-msoft-float or -mhard-float	Generates code that does not use (uses) the floating-point register set. Software floating point emulation is provided if you use the -msoft-float option, and passes the option to GCC when linking.

Option	Description
-mmultiple or -mno-multiple	Generates code that uses (does not use) the load multiple word instructions and the store multiple word instructions. These instructions are generated by default on POWER systems, and not generated on PowerPC systems. Do not use -mmultiple on little endian PowerPC systems, since those instructions do not work when the processor is in little endian mode. The exceptions are PPC740 and PPC750, which permit the instructions usage in little endian mode.
-mstring or -mno-string	Generates code that uses (does not use) the load string instructions and the store string word instructions to save multiple registers and do small block moves. These instructions are generated by default on POWER systems, and not generated on PowerPC systems. Do not use -mstring on little endian PowerPC systems, since those instructions do not work when the processor is in little endian mode. The exceptions are PPC740 and PPC750, which permit the instructions usage in little endian mode.
-mupdate or -mno-update	Generates code that uses (does not use) the load or store instructions that update the base register to the address of the calculated memory location. These instructions are generated by default. If you use -mno-update, there is a small window between the time that the stack pointer is updated and the address of the previous frame is stored, which means code that walks the stack frame across interrupts or signals may get corrupted data.
-mfused-madd or -mno-fused-madd	Generates code that uses (does not use) the floating point multiply and accumulate instructions. These instructions are generated by default if hardware floating is used.
-mno-bit-align or -mbit-align	On System V.4 and embedded PowerPC systems, does not (does) force structures and unions that contain bit fields to be aligned to the base type of the bit field. For example, a structure containing nothing but eight unsigned bitfields of length 1 would be, by default, aligned to a four byte boundary and have a size of 4 bytes. By using -mno-bit-align, the structure would be aligned to a one byte boundary and be one byte in size.
-mno-strict-align or -mstrict-align	On System V.4 and embedded PowerPC systems, does not (does) assume that unaligned memory references will be handled by the system.
-mrelocatable or -mno-relocatable	On embedded PowerPC systems, generates code that allows (does not allow) the program to be relocated to a different address at run time. If you use -mrelocatable on any module, all objects linked together must be compiled with -mrelocatable or -mrelocatable-lib.
-mrelocatable-lib or -mno-relocatable-lib	On embedded PowerPC systems, generates code that allows (does not allow) the program to be relocated to a different address at run time. Modules compiled with -mrelocatable-lib can be linked with either modules compiled without -mrelocatable and -mrelocatable-lib or with modules compiled with the -mrelocatable options.

Option	Description
-mno-toc or -mtoc	On System V.4 and embedded PowerPC systems, does not (does) assume that register 2 contains a pointer to a global area pointing to the addresses used in the program.
-mlittle or -mlittle-endian	On System V.4 and embedded PowerPC systems, compiles code for the processor in little endian mode. The -mlittle-endian option is the same as -mlittle.
-mbig or -mbig-endian	On System V.4 and embedded PowerPC systems, compiles code for the processor in big endian mode. The -mbig-endian option is the same as -mbig.
-mcall-sysv	On System V.4 and embedded PowerPC systems, compiles code using calling conventions that adheres to the March 1995 draft of the System V Application Binary Interface, PowerPC processor supplement. This is the default unless you configured GCC using powerpc-*-eabiaix.
-mcall-sysv-eabi	Specifies both -mcall-sysv and -meabi options.
-mcall-sysv-noeabi	Specifies both -mcall-sysv and -mno-eabi options.
-mcall-aix	On System V.4 and embedded PowerPC systems, compiles code using calling conventions that are similar to those used on AIX. This is the default if you configured GCC using powerpc-*-eabiaix.
-mcall-solaris	On System V.4 and embedded PowerPC systems, compiles code for the Solaris operating system.
-mcall-linux	On System V.4 and embedded PowerPC systems, compiles code for the Linux-based GNU system.
-mprototype or -mno-prototype	On System V.4 and embedded PowerPC systems, assumes that all calls to variable argument functions are properly prototyped. Otherwise, the compiler must insert an instruction before every non prototyped call to set or clear bit 6 of the condition code register (CR) to indicate whether floating point values were passed in the floating point registers in case the function takes a variable arguments. With -mprototype, only calls to prototyped variable argument functions will set or clear the bit.
-msim	On embedded PowerPC systems, assumes that the startup module is called sim-crt0.o and that the standard C libraries are libsim.a and libc.a. This is the default for powerpc-*-eabisim configurations.
-mmvme	On embedded PowerPC systems, assumes that the startup module is called crt0.o and the standard C libraries are libmvme.a and libc.a.
-mads	On embedded PowerPC systems, assumes that the startup module is called crt0.o and the standard C libraries are libads.a and libc.a.
-myellowknife	On embedded PowerPC systems, assumes that the startup module is called crt0.o and the standard C libraries are libyk.a and libc.a.

Option	Description
-mvxworks	On System V.4 and embedded PowerPC systems, specifies that you are compiling for a VxWorks system.
-memb	On embedded PowerPC systems, sets the PPC_EMB bit in the ELF flags header to indicate that eabi extended relocations are used.
-meabi or -mno-eabi	On System V.4 and embedded PowerPC systems, does (does not) adhere to the Embedded Applications Binary Interface (eabi) which is a set of modifications to the System V.4 specifications. Selecting @option{-meabi means that the stack is aligned to an eight byte boundary, a function __eabi is called to from main to set up the eabi environment, and the -msdata option can use both r2 and r13 to point to two separate small data areas. Selecting @option{-mno-eabi means that the stack is aligned to a 16 byte boundary, does not call an initialization function from main, and the -msdata option will only use r13 to point to a single small data area. The -meabi option is on by default if you configured GCC using one of the powerpc*-*-eabi* options.
-msdata=eabi	On System V.4 and embedded PowerPC systems, puts small initialized const global and static data in the .sdata2 section, which is pointed to by register r2. Puts small initialized non-const global and static data in the .sdata section, which is pointed to by register r13. Put small uninitialized global and static data in the .sbss section, which is adjacent to the .sdata section. The -msdata=eabi option is incompatible with the -mrelocatable option. The -msdata=eabi option also sets the -memb option.
-msdata=sysv	On System V.4 and embedded PowerPC systems, puts small global and static data in the .sdata section, which is pointed to by register r13. Put small uninitialized global and static data in the .sbss section, which is adjacent to the .sdata section. The -msdata=sysv option is incompatible with the -mrelocatable option.
-msdata=default or -msdata	On System V.4 and embedded PowerPC systems (if -meabi is used), compiles code the same as -msdata=eabi, otherwise compiles code the same as -msdata=sysv.
-msdata-data	On System V.4 and embedded PowerPC systems, puts small global and static data in the .sdata section. Put small uninitialized global and static data in the .sbss section. Does not use register r13 to address small data. This is the default behavior unless other -msdata options are used.
-msdata=none or -mno-sdata	On embedded PowerPC systems, puts all initialized global and static data in the .data section, and all uninitialized data in the .bss section.
-G num	On embedded PowerPC systems, puts global and static items less than or equal to num bytes into the small data or bss sections instead of the normal data or bss section. By default, num is 8. The -G num switch is also passed to the linker. All modules should be compiled with the same -G num value.

Option	Description
-mregnames or -mno-regnames	On System V.4 and embedded PowerPC systems, does (does not) emit register names in the assembly language output using symbolic forms.

C.14.3 Flags specific to Intel Itanium-based systems

The -m options defined for Intel Itanium-based systems are shown in Table 145.

Table 145. Itanium specific options for GNU GCC

Option	Description
-mbig-endian	Generates code for a big endian target. This is the default for HP-UX.
-mlittle-endian	Generates code for a little endian target. This is the default for AIX 5L and Linux.
-mgnu-as -mno-gnu-as	Generates (or does not generate) code for the GNU assembler. This is the default.
-mgnu-ld -mno-gnu-ld	Generates (or does not generate) code for the GNU linker. This is the default.
-mno-pic	Generates code that does not use a global pointer register. The result is not position independent code, and violates the IA-64 ABI.
-mvolatile-asm-stop -mno-volatile-asm-stop	Generates (or does not generate) a stop bit immediately before and after volatile asm statements.
-mb-step	Generates code that works around Itanium B step errata.
-mregister-names -mno-register-names	Generates (or does not generate) in, loc, and out register names for the stacked registers. This may make assembler output more readable.
-mno-sdata -msdata	Disables (or enables) optimizations that use the small data section. This may be useful for working around optimizer bugs.
-mconstant-gp	Generates code that uses a single constant global pointer value. This is useful when compiling kernel code.
-mauto-pic	Generates code that is self-relocatable. This implies -mconstant-gp. This is useful when compiling firmware code.
-minline-divide-min-latency	Generates code for inline divides using the minimum latency algorithm.
-minline-divide-max-throughput	Generates code for inline divides using the maximum throughput algorithm.
-mno-dwarf2-asm -mdwarf2-asm	Does (or does not) generate assembler code for the DWARF2 line number debugging info. This may be useful when not using the GNU assembler.

Option	Description
-mfixed-range =register range	Generates code treating the given register range as fixed registers. A fixed register is one that the register allocator cannot use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

Appendix D. Using the additional material

The source code examples used in this redbook are available for download from the Web.

D.1 Locating the additional material on the Internet

The source code examples associated with this redbook are available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246034/

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

D.2 Using the Web material

The additional Web material that accompanies this redbook includes the following:

File name	Description
6034samples.tar.Z	Compressed tar archive of source code samples.

D.2.1 System requirements for downloading the Web material

The following system configuration is recommended for downloading the additional Web material.

Hard disk space:	1 MB minimum

D.2.2 How to use the Web material

Create a subdirectory (folder) on your workstation and copy the contents of the Web material into this folder. Extract the source code examples with the following command:

```
zcat 6034samples.tar.Z | tar xvf -
```

Use the sample code to examine the features of AIX 5L.

The contents of the 6034samples.tar archive are listed below.

```
# zcat 6034samples.tar.Z | tar tvf -
-rwxr-xr-x   0 1     1025 Jun 04 21:38:59 2001 find_internal_macro_aix.ksh
-rwxr-xr-x   0 1     1145 Jun 04 21:36:26 2001 find_predef_macro_aix.ksh
-rwxr-xr-x   0 1     1010 Jun 04 21:36:35 2001 find_predef_macro_gnu.ksh
-rwxr-xr-x   0 1     1130 Jun 04 21:36:41 2001 find_spec_targets_aix.ksh
-rwxr-xr-x   0 1     1124 Jun 04 21:36:52 2001 find_spec_targets_gnu.ksh
-rwxr-xr-x   0 1     1721 Jun 04 21:49:27 2001 cpu_bind.c
-rwxr-xr-x   0 1     3985 Jun 04 21:41:07 2001 hwinfo.c
-rwxr-xr-x   0 1     5325 Jun 04 21:42:28 2001 mandelbrot1.c
-rwxr-xr-x   0 1     2477 Jun 04 21:43:34 2001 mandelbrot2.c
-rwxr-xr-x   0 1     2995 Jun 04 21:44:52 2001 mandelbrot3.c
-rwxr-xr-x   0 1     4528 Jun 04 21:46:08 2001 mandelbrot4.c
-rwxr-xr-x   0 1     7415 Jun 04 21:47:52 2001 mandelbrot5.c
-rwxr-xr-x   0 1     1047 Jun 04 21:50:27 2001 mem_lock.c
-rwxr-xr-x   0 1     5198 Jun 04 21:43:54 2001 message.c
-rwxr-xr-x   0 1     9976 Jun 04 21:44:21 2001 semaphore1.c
-rwxr-xr-x   0 1     4742 Jun 04 21:52:17 2001 shared_mem.c
-rwxr-xr-x   0 1     6506 Jun 04 21:53:57 2001 signals1.c
-rwxr-xr-x   0 1     6841 Jun 04 21:56:33 2001 signals2.c
-rwxr-xr-x   0 1     3091 Jun 04 21:46:57 2001 timer.c
```

Appendix E. Special notices

This publication is intended to help application developers port their applications to the AIX 5L operating system. The information in this publication is not intended as the specification of any programming interfaces that are provided by the AIX 5L operating system. See the PUBLICATIONS section of the IBM Programming Announcement for the AIX 5L operating system for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers

AIX	Approach
AS/400	DB2
e (logo)	IBM ®
Lotus	Notes
Open Class	PartnerWorld
PowerPC	Redbooks
Redbooks Logo	RMF
RS/6000	SAA
SP	System/390
VisualAge	XT

Appendix F. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

F.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 601.

- *AIX 5L Workload Manager (WLM)*, SG24-5977
- *C and C++ Application Development on AIX*, SG24-5674
- *IBM Certification Study Guide AIX Installation and System Recovery*, SG24-6183
- *Running Linux Applications on AIX*, SG24-6033

F.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at ibm.com/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	**Collection Kit Number**
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

F.3 Other resources

These publications are also relevant as further information sources of information. They are subdivided by topic.

F.3.1 C and C++ language

- Ellis, et al., *The Annotated C++ Reference Manual*, Addison Wesley Longman, Inc., 1990, ISBN 0201514591
- Kernighan, et al., *The C Programming Language, Second Edition*, Prentice-Hall PTR, 1989. ISBN 0131103628
- Lippman, et al., *C++ Primer*, Third Edition, Addison Wesley Longman, Inc., 1998, ISBN 0201824701
- Stroustrup, et al., *The Design and Evolution of C++*, Addison Wesley Longman, Inc., 1994, ISBN 0201543303
- *Programming Languages - C* (ANSI/ISO/IEC 9899-1999), found at:

  `http://web.ansi.org/public/std_info.html`
- *Programming Languages - C++* (ANSI/ISO/IEC 14882-1998) found at:

  `http://web.ansi.org/public/std_info.html`

F.3.2 C and C++ Development on AIX

- *AIX 5L Online Documentation - General Programming Concepts: Writing and Debugging Programs*
- *AIX 5L Online Documentation - System Management Concepts: Operating System and Devices*
- *AIX 5L Online Documentation - System Management Guide: Operating System and Devices*
- *AIX 5L Online Documentation - System's User Guide: Operating System and Devices*

These AIX 5L documents can be viewed online at:

`http://www-1.ibm.com/servers/aix/library/index.html`

under the section Technical Publications. The documentation is currently available online in the following languages: English, Japanese, Korean, Spanish, German, Portuguese, French, Traditional Chinese and Italian.

- *C for AIX User's Guide* - This can only be found in the AIX C compiler online documentation.
- *Using License Use Management Guide Runtime for AIX*, SH19-4346, found at:

  `ftp://ftp.software.ibm.com/software/lum/aix/doc/V4.5.5/lumusgaix.pdf`

F.3.3 VisualAge C++ and C for AIX compilers

The support page for the VisualAge C++ Professional for AIX Version 5 compiler is:

`http://www.ibm.com/software/ad/vacpp/support.html`

The compiler documentation can be downloaded in PDF format from the following URLs:

`http://www.ibm.com/software/ad/vacpp/library.html`
`http://www.ibm.com/software/ad/vacpp/support.html`

The support page for the C for AIX Version 5 compiler is:

`http://www.ibm.com/software/ad/caix/support.html`

AIX developer information, including white papers, sample code, and technology articles, can be located on the Web at the following URL:

`http://www.developer.ibm.com`

F.3.4 Threads

The following references contain additional information on threads.

- IEEE Standards Online Subscriptions, found at:

  `http://standards.ieee.org/catalog/olis/index.html`

  All IEEE Standards referred to in this book may be found there. This Web site requires registration, which can be done online. Use search criteria, such as 9945-1 or 1003.1.
- *The Single UNIX Specification, Version 2*, found at:

  `http://www.opengroup.org/onlinepubs/7908799/toc.htm`
- Lewine, et al., *Posix Programmer's Guide: Writing Portable Unix Programs With the Posix.1 Standard*, O'Reilly & Associates, Inc., 1992, ISBN 0937175730.

F.3.5 Standards

Standards can be found at:

- `http://standards.ieee.org`

F.4 Web sites

The following Web sites may be useful sources of information:

- `www.ibm.com/servers/eserver/linux` - Linux for IBM @server Web page
- `http://www-1.ibm.com/servers/aix/library/index.html` - IBM AIX Library Web page
- `www.kornshell.com` - KornShell Web page
- `www.ibm.com/servers/aix/products/aixos/linux/` - AIX Toolbox for Linux Applications Web page
- `http://www-frec.bull.com/pub` - Bull's Large Freeware and Shareware Archive for AIX Web page
- `http://ftp.univie.ac.at/aix` - Bull's Large Freeware and Shareware Archive for AIX mirror site
- `www.rge.com/pub/systems/aix/bull` - Bull's Large Freeware and Shareware Archive for AIX mirror site
- `ftp://ftp.rge.com/pub/systems/aix/bull` - Bull's Large Freeware and Shareware Archive for AIX mirror site
- `www.gnu.org` - GNU home page
- `www.au.nedit.org` - Nirvana Editor Web site

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** **ibm.com**/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	**Order Number**	**Quantity**

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Abbreviations and acronyms

ANSI	American National Standards Institute
API	Application Programming Interface
CD	Compact Disc
CD-ROM	Compact Disc-Read Only Memory
CMA	Concert Multithread Architecture
CPU	Central Processing Unit
DCE	Distributed Computing Environment
HTTP	Hypertext Transfer Protocol
IA	Intel Architecture
IBM	International Business Machines Corporation
ILP32	Integer Long Pointer 32
I/O	Input/Output
IPC	Inter-Process Communication
ITSO	International Technical Support Organization
LP64	Long Pointer 64
LPP	Licensed Program Product
MP	Multiprocessor
NFS	Network File System
PCI	Peripheral Component Interconnect
POSIX	Portable Operating Interface for Computing Environments
POWER	Performance Optimization with Enhanced RISC (Architecture)
SMIT	System Management Interface Tool
SMP	Symmetric Multiprocessor
TCP/IP	Transmission Control Protocol/Internet Protocol
TSD	Thread Specific Data
UP	Uniprocessor
VA	VisualAge
VMM	Virtual Memory Manager

Index

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-6034-00 AIX 5L Porting Guide
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	 O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

IBM

Redbooks

AIX 5L Porting Guide

AIX 5L Porting Guide

Practical advice and guidance when porting to AIX 5L

Common problems explained and solutions documented

Written by developers for developers

This redbook details the types of problems most likely to be encountered when porting applications from other UNIX-based platforms to the AIX 5L Operating System.

When porting an application to a new operating system there are things you have to know and questions you have to ask, such as:

- What programming models are available?
- How are threads implemented?
- What link options do I need?
- Why do my makefiles not work any more?

We have tried to condense all of these questions (and answers) into one document, and this redbook is the result. It has been designed to provide guidance and reference material for system and application programmers who have been given the task of porting applications written in C and/or C++ to the AIX 5L operating system. This redbook assumes the reader is familiar with the C and/or C++ programming languages and UNIX operating systems.

SG24-6034-00 ISBN 0738422053