POSIX.1c Migration Guide

LynxOS Release 4

DOC-0415-00

Product names mentioned in *POSIX.1c Migration Guide* are trademarks of their respective manufacturers and are used here for identification purposes only.

Printed in the United States of America.

Contents

Preface

The *POSIX 1.c Migration Guide* is intended to help developers migrate code developed under the POSIX.4a Draft 4 Pthreads extensions to the ISO/IEC 9945-1:1996 (POSIX.1) Threads Options, also known as POSIX.1c. This guide is not meant as instruction to write code directly to the POSIX.1c standard. This guide also assumes that the reader is thoroughly familiar with POSIX.4a Draft 4. For detailed information, consult the appropriate LynxOS man pages and the ISO/IEC 9945-1:1996 standard.

For More Information

For more information on the features of LynxOS, refer to the following printed and online documentation.

- *Release Notes*

  This printed document contains late-breaking information about the current release.

- *LynxOS Installation Guide*

  This manual supports the initial installation and configuration of LynxOS and the X Windows System.

- *LynxOS User's Guide*

  This document contains information about basic system administration and kernel-level specifics of LynxOS. It contains a "Quick Starting" chapter and covers a range of topics, including tuning system performance and creating kernel images for embedded applications.

- Online information

  Information about commands and utilities is provided online in text format through the **man** command. For example, a user wanting

information about the GNU compiler would enter the following syntax, where **gcc** is the argument for information about the GNU compiler:

```
man gcc
```

More recent versions of the documentation listed here may also be found online.

Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to file names and commands are case sensitive and should be typed accurately.

Kind of Text	Examples
Body text; *italicized* for emphasis, new terms, and book titles	Refer to the *LynxOS User's Guide.*

Environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data
Commands that need to be highlighted within body text, or commands that must be typed as is by the user are **bolded**.
	`ls`
	`-l`
	`myprog.c`
	`/dev/null`
	`login: myname`
	`# cd /usr/home`

Text that represents a variable, such as a file name or a value that must be entered by the user	`cat filename` `mv file1 file2`

Kind of Text	Examples

Blocks of text that appear on the display screen after entering instructions or commands	`Loading file /tftpboot/shell.kdi into 0x4000`
	`.....`
	`File loaded. Size is 1314816`
	`Copyright 2000 LynuxWorks, Inc.`
	`All rights reserved.`
	
	`LynxOS (ppc) created Mon Jul 17 17:50:22 GMT 2000`
	`user name:`

Keyboard options, button names, and menu sequences	**Enter**, **Ctrl-C**

Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

NOTE: These callouts note important or useful points in the text.

CAUTION! Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

Technical Support

LynuxWorks Technical Support is available Monday through Friday (holidays excluded) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

The LynuxWorks World Wide Web home page provides additional information about our products, Frequently Asked Questions (FAQs), and LynuxWorks news groups.

LynuxWorks U.S. Headquarters

Internet: support@lnxw.com
Phone: (408) 979-3940
Fax: (408) 979-3945

LynuxWorks Europe

Internet: tech_europe@lnxw.com
Phone: (+33) 1 30 85 06 00
Fax: (+33) 1 30 85 06 06

World Wide Web

http://www.lynuxworks.com

CHAPTER 1 *Introduction*

POSIX.1c Description

The POSIX.1c standard encompasses the threads option to the POSIX.1standard. POSIX.1c functionality includes thread management, scheduling, and cancellation, thread-specific data, fork, signal handling, mutexes, and condition variables.

This guide describes the differences between compiler flags in POSIX.4a Draft 4 and POSIX.1c.

For a guide to POSIX.1c programming style with LynxOS, please refer to the online example programs provided with the distribution and other documents.

The following definitions clarify how the various POSIX specifications interrelate.

POSIX.1 — This is the basic operating system standard, also known as POSIX 1003.1. The current version of POSIX.1 was approved in 1996, and encompasses the 1988 version, as well as the final versions of POSIX.1b and POSIX.1c.

POSIX.1b — This contains amendments to POSIX.1 for real-time systems. POSIX.1b was approved in 1993. It is also known as POSIX.4 Draft 14.

POSIX.1c (previously POSIX.4a)

Contains amendments to POSIX.1 defining thread primitives - thread creation, synchronization, scheduling, etc. The POSIX.4 committee decided that keeping threads in the POSIX.1b standard would delay approval, due to the complexity of the threads issue. Also, a separate POSIX specification for the threads interface allows vendors the option to exclude the other real-time support that POSIX.1b requires. POSIX.4a Draft 8 was approved and renamed POSIX.1c.

LynxOS is compliant with the ISO/IEC 9945-1:1996 POSIX.1c standard.

In this guide, the term *POSIX.1c* refers to the Threads Option of the POSIX.1 standard.

NOTE: Throughout the code examples in this document, error checking is not performed; function calls for which error checking is not executed are assumed to be successful. The reader should not accept this as a programming style. Error checking is omitted in order to keep this document concise and emphasize migration. In order to simplify the examples, obvious include files (e.g., `<stdio.h>`) have not been shown.

Overview of Major Changes

- The compilation environment now defaults to POSIX.1c, rather than POSIX.4a. To invoke POSIX.4a functionality, you must specify the option **`-mthreads-pre1c`** when compiling your application.
- Significant library structure changes have been made in this release of LynxOS. Please see “Library Structure & Compiler Option Changes” on page 3 for more detailed information.
- It is not possible to mix the POSIX.4a and POSIX.1c standards in the same application. You must choose between the two with compile-time options. LynuxWorks recommends using the POSIX.1c standard for all future development.
- Not all POSIX.4a features have corresponding equivalents in POSIX.1c. Some features have been discontinued.
- There are new facilities in POSIX.1c not contained in POSIX.4a. For example, threads may be created in a detached state and can register fork handlers; mutexes and condition variables now have attributes and may be statically initialized.
- Error reporting has been significantly revised. In POSIX.4a, `pthreads` library functions return `0` for success, or `-1` to indicate an error and set the `errno` global to the specific error number. In POSIX.1c, most functions return `0` for success or the error number directly.

Library Structure & Compiler Option Changes

In this release of LynxOS, both POSIX.1c and POSIX.4a are supported. However, POSIX.1c is now the default compilation environment. This change has involved modifying the library structure and compile time options in LynxOS.

Library Routiness

All POSIX.1 library routines, including those for POSIX.1c, are merged into the library `libc.a`. This library exists in `/lib` and `/lib/thread` (for multithreads).

LynxOS includes a backward-compatibility library:

`libposix-pre1c.a` Contains support for POSIX.4a

Compiler Options Changes

POSIX.1c is the default for LynxOS. To compile an application with POSIX.4a functionality, you must compile with the **`-mlegacy-threads`** option. This option signals the compiler to define **`-D_THREADS_POSIX4ad4`** and to link with the `libposix-pre1c.a` library. Thus, you do not need to change POSIX.4a source code, but you do need to recompile with the `-mlegacy-threads` option.

Also, you cannot mix POSIX.4a functionality with POSIX.1c in a single application. Your application may fail to link or may exhibit unpredictable results at run-time.

The following table summarizes the changes to the compiler options.

Table 1-1: Compiler Option Changes in LynxOS 4.0

POSIX Specification	LynxOS gcc
POSIX 1003.1 C Language Standard	Default
POSIX 1003.1b Final version of 1003.4	Default
POSIX 1003.4a Draft 4 Thread Extensions	`-mlegacy-threads`
POSIX.1c Final standard	`-mthreads`

NOTE: If you specify the option `-mposix` when compiling an application with either `cc` (x86 only) or `gcc`, the compiler displays an error message and aborts the compile. *You must change compiler directives to invoke the options as listed in the table above.*

Updating Applications

If you previously used **`-mthreads`** to compile applications, there are two alternatives for updating to the new structure:

1. Make code POSIX.1c-compliant.
2. If compiling POSIX.4a-compliant code, change all `-mthreads` compilation switches to **`-mlegacy-threads`**.

Here are some simple shell scripts you can use to make these needed changes to your environment. The scripts assume that `bash` is the current shell.

To change `-mthreads` to `-mlegacy-threads` in all Makefiles named `Makefile`:

```
bash$ for f in `find . -name Makefile -print`
> do
> cp $f $f.orig
> sed 's/-mthreads/-mlegacy-threads/' $f.orig > $f
> done
```

Other General Changes

Refer to the **`sysconf()`** man page for a list of new parameters that can be passed to the `sysconf()` function. Also, refer to the **`pathconf()`** and **`fpathconf()`** man pages for a list of new parameters that can be passed to these two functions.

The lists of run-time invariant values and compile-time symbolic constants have changed. Due to their length, these lists are not included here. Refer to Table 2.5 and Table 2.10 of the POSIX.1 specification (ISO/IEC 9945-1:1996) for these lists.

CHAPTER 2 *Thread Management*

Thread Creation Attributes

POSIX.1c changes to thread creation attributes are relatively minor. The following table is a comparison of the revised interfaces and their corresponding POSIX.4a versions:

Table 2-1: Thread Creation Attribute Function Changes

POSIX.1c	POSIX.4a
`int pthread_attr_init( pthread_attr_t *attr)`	`int pthread_attr_create( pthread_attr_t *attr)`
`int pthread_attr_destroy( pthread_attr_t *attr)`	`int pthread_attr_delete( pthread_attr_t *attr)`
`int pthread_attr_setstacksize( pthread_attr_t *attr, size_t stacksize)`	`int pthread_attr_setstacksize( pthread_attr_t *attr, long stacksize)`
`int pthread_attr_getstacksize (pthread_attr_t *attr,size_t *stacksize)`	`unsigned long pthread_attr_getstacksize (pthread_attr_t *attr)`

Table 2-1: Thread Creation Attribute Function Changes (Continued)

POSIX.1c	POSIX.4a
`int pthread_attr_setdetachstate (pthread_attr_t *attr,int detachstate)`	No equivalent
`int pthread_attr_getdetachstate (pthread_attr_t *attr,int *detachstate)`	No equivalent

In LynxOS, the POSIX.1c function **`pthread_attr_destroy()`** sets the `attribute` object to an invalid value to prevent its subsequent use without reinitialization.

POSIX.4a does not provide a facility to create a thread in anything other than a joinable state, requiring a thread to be explicitly detached via **`pthread_detach()`**. POSIX.1c allows a thread to be created in a detached state by using **`pthread_attr_setdetachstate()`** to set the **`detachstate`** attribute to `PTHREAD_CREATE_DETACHED` (the default is `PTHREAD_CREATE_JOINABLE`).

Thread Management Functions

POSIX.1c changes to thread management are also relatively minor. The following table is a comparison of the revised interfaces and their corresponding POSIX.4a versions:

Table 2-2: Thread Management Function Changes

POSIX.1c	POSIX.4a
`int pthread_create (pthread_t *thread,pthread_attr_t *attr,void *(*start_routine) void *), void *arg)`	`int pthread_create (pthread_t *thread, pthread_attr_t attr, void *(*start_routine) (void *), void *arg)`
`int pthread_detach( pthread_t thread)`	`int pthread_detach (pthread_t *thread)`

Thread Fork Handlers

POSIX.1c defines the **`pthead_atfork()`** function to register handler functions to be called before and after **`fork()`**, in the context of the thread that called `fork()`. POSIX.4a has no such facility.

Dynamic Package Initialization

In POSIX.4a the constant **`pthread_once_init`** (all lowercase letters) is defined as the initializer for a `once` control object used with **`pthread_once()`**, as follows:

```
pthread_once_t once_block = pthread_once_init;
```

POSIX.1c defines the constant **`PTHREAD_ONCE_INIT`** (all uppercase letters) for this purpose:

```
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

CHAPTER 3 *Thread Scheduling*

Thread Creation Scheduling Attributes

The thread creation scheduling attributes functions and definitions have changed considerably in POSIX.1c. The following table is a comparison of the revised interfaces and their corresponding POSIX.4a versions:

Table 3-1: Thread Creation Scheduling Interface Changes

POSIX.1c	POSIX.4a
`int pthread_attr_setscope (pthread_attr_t *attr, int contentionscope)`	No equivalent
`int pthread_attr_getscope (pthread_attr_t *attr, int *contentionscope)`	No equivalent
`int pthread_attr_setinheritsched (pthread_attr_t *attr, int inheritsched);`	`int pthread_attr_setinheritsched (pthread_attr_t attr)`
`int pthread_attr_getinheritsched (pthread_attr_t *attr, int *inheritsched)`	`int pthread_attr_getinheritsched (pthread_attr_t attr)`
`int pthread_attr_setschedpolicy (pthread_attr_t *attr,int policy)`	`int pthread_attr_setsched (pthread_attr_t *attr,int scheduler)`

Table 3-1: Thread Creation Scheduling Interface Changes (Continued)

POSIX.1c	POSIX.4a
`int pthread_attr_getschedpolicy (pthread_attr_t *attr,int *policy)`	`int pthread_attr_getsched (pthread_attr_t attr)`
`int pthread_attr_setschedparam (pthread_attr_t *attr,const struct sched_param *param)`	`int pthread_attr_setprio (pthread_attr_t *attr, int priority)`
`int pthread_attr_getschedparam (pthread_attr_t *attr, struct sched_param *param)`	`int pthread_attr_getprio (pthread_attr_t attr)`
`PTHREAD_SCOPE_SYSTEM`	No equivalent
`PTHREAD_SCOPE_PROCESS`	No equivalent
`PTHREAD_EXPLICIT_SCHED`	`PTHREAD_DEFAULT_SCHED`
`sched_get_priority_min()`	`PRIO_???_MIN macros`
`sched_get_priority_max()`	`PRIO_???_MAX macros`

POSIX.4a does not directly support the concept of thread scheduling scope. This facility is provided in POSIX.1c by the **`pthread_attr_setscope()`** and **`pthread_attr_getscope()`** functions, and the associated **`PTHREAD_SCOPE_SYSTEM`** and **`PTHREAD_SCOPE_PROCESS`** symbolic constants.

NOTE: LynxOS supports only the default system thread scheduling scope (`PTHREAD_SCOPE_SYSTEM`). Attempts to set the scheduling scope to `PTHREAD_SCOPE_PROCESS` returns an `ENOTSUP` error.

The scheduling policy values (**`SCHED_FIFO`**, **`SCHED_RR`**, **`SCHED_OTHER`**) are unchanged.

In addition to these scheduling policies, LynxOS implements non-preemptible scheduling policy SCHED_NONPREEMPT. See “Non-Preemptible Scheduling Policy” on page 13 for a detailed description of this policy.

In POSIX.1c, a thread’s scheduling priority attribute is set or retrieved using a **`sched_param`** structure, instead of specifying or retrieving it directly.

POSIX.4a uses the POSIX.4 Draft 9 scheduling minimum and maximum priority macros, such as **PRIO_RR_MIN** or **PRIO_OTHER_MAX**, for thread priority limits for each scheduling policy. These are replaced in POSIX.1c with the use of the POSIX.1b scheduling parameter limits retrieval functions as shown in the above table (see ISO/IEC 9945-1 1996 Section 13.3.6).

The following code fragments illustrate the differences in manipulating thread priority attributes between POSIX.4a and POSIX.1c:

POSIX.4a Code

```
main()
{
   int rv;
   int prio;
   pthread_attr_t attr;
   :
   /* Set the initial priority for the new thread */
   prio = PRIO_OTHER_MIN + 10;
   rv = pthread_attr_setprio(&attr, prio);
   :
}
```

Equivalent POSIX.1c Code

```
main()
{
   int rv;
   int prio;
   struct sched_param param;
   pthread_attr_t attr;
   :
   /* Set the initial priority for the new thread */
   prio = sched_get_priority_min(SCHED_OTHER);
   param.sched_priority = prio + 10;
   rv = pthread_attr_setschedparam(&attr, &param);
   :
}
```

Non-Preemptible Scheduling Policy

LynxOS implements a non-preemptible scheduling policy called `SCHED_NONPREEMPT`. A process running under this policy cannot be preempted by any other process until it voluntarily sleeps or blocks waiting for a semaphore or mutex object. An example of this policy is a garbage collector running with low

priority and performing critical work so that it must not be interrupted. The sample code of the application is as follows:

```
void garbage_collector(arg)
void *arg;
{
  ...
  while (1) { /* endless loop */
    if (waste_ratio() > max_waste_ratio) {
      /* garbage collection, critical area
       */
      ...
    }
    sleep(GC_TIMEOUT);
  } /* end of loop */
}

main()
{
  pthread_attr_t attr;
  struct sched_param prio;
  pthread_t tid;
  ...
  pthread_attr_create(&attr);
  pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
  pthread_attr_setschedpolicy(&attr, SCHED_NONPREEMPT);
  prio.sched_priority = PRIO_NONPREEMPT_MIN;
  pthread_attr_setschedparam(&attr, &prio);
  pthread_create(&tid, &attr, garbage_collector, NULL);
  ...
}
```

LynxOS also defines 2 constants for the `SCHED_NONPREEMPT` policy designating the maximum and minimum priorities for this policy. These constants are:

- `PRIO_NONPREEMPT_MAX`
- `PRIO_NONPREEMPT_MIN`

These priorities can also be obtained using the `sched_get_priority_max()` and `sched_get_priority_min()` functions.

Dynamic Thread Scheduling Parameters Access

The POSIX.1c thread scheduling parameter access functions are not one-to-one replacements for their POSIX.4a counterparts, as shown in the following table:

Table 3-2: Thread Scheduling Parameter Access Function Changes

POSIX.1c	POSIX.4a

`int pthread_setschedparam (pthread_t thread,int policy,const struct sched_param *param)`	`int pthread_setscheduler (pthread_t thread, int alg,int prio)`
	
	`int pthread_setprio (pthread_t thread, int prio)`

`int pthread_getschedparam (pthread_t thread,int *policy,struct sched_param *param)`	`int pthread_getscheduler (pthread_t thread)`
	
	`int pthread_getprio pthread_t thread)`

Each of the POSIX.1c functions combines the functionality of two corresponding POSIX.4a functions. The following code fragments illustrate the usage differences:

POSIX.4a Code

```
main ()
{
   int rc;
   pthread_t thread;
   int prio;
   :
   /* Set the policy and priority of the thread */
   prio = PRIO_FIFO_MIN + 10;
   rc = pthread_setscheduler(thread, SCHED_FIFO, prio);
   :
   /* Now change only the thread's priority */
   prio = pthread_getprio(thread);
   rc = pthread_setprio(thread, prio + 5);
   :
}
```

POSIX.1c Code

```
main()
{
   int rc;
   pthread_t thread;
   int policy = SCHED_FIFO;
   struct sched_param param;
   :
   /* Set the policy and priority of the thread */
   prio = sched_get_priority_min(policy);
   param.sched_priority = prio + 10;
   rc = pthread_setschedparam(thread, policy, &param);
   :
   /* Now change only the thread's priority */
   rc = pthread_getschedparam(thread, &policy, &param);
   param_sched_priority += 5;
   rc = pthread_setschedparam(thread, policy, &param);
   :
}
```

As in "Thread Creation Scheduling Attributes" on page 11, the use of the POSIX.4 Draft 9 scheduling minimum and maximum priority macros in POSIX.4a are replaced by the POSIX.1b scheduling parameter limits retrieval functions in POSIX.1c.

Relinquishing the Processor

The POSIX.4a **`pthread_yield()`** function has been replaced in POSIX.1c by the POSIX.1b **`sched_yield()`** function.

Calls to `sched_yield()` on LynxOS always return `0`.

CHAPTER 4 *Thread-Specific Data*

Version Variations

POSIX.1c incorporates minor revisions to the thread-specific data-related functions versus the POSIX.4a interfaces, as shown in the following table:

Table 4-1: Thread-Specific Data Function Changes

POSIX.1c	POSIX.4a
`int pthread_key_create (pthread_key_t *key,void (*destructor)void *value))`	`int pthread_keycreate (pthread_key_t *key,void (*destructor)(void *value))`
`int pthread_key_delete (pthread_key_t key)`	No equivalent
`int pthread_setspecific (pthread_key_t key,const void *value)`	`int pthread_setspecific (pthread_key_t key,void *value)`
`void * pthread_getspecific (pthread_key_t key)`	`int pthread_getspecific (pthread_key_t key,void **value)`

POSIX.4a does not provide a facility for deleting a thread-specific data key, which is provided in POSIX.1c by the **`pthread_key_delete()`** function.

NOTE: The `pthread_key_delete()` function in LynxOS is currently a no-op. Therefore, repeatedly deleting and recreating keys in a thread may cause the per-thread key limit to be exceeded.

In POSIX.1c, the **`pthread_getspecific()`** function cannot return an error. If no thread-specific data value is associated with the specified key, `pthread_getspecific()` returns `NULL`.

NOTE: This also means that setting the value associated with a key to `NULL` using `pthread_setspecific()` results in an ambiguous return value from a subsequent call to `pthread_getspecific()` for that key.

CHAPTER 5 *Thread Cancellation*

Thread Cancellation Control

The thread cancellation control interfaces have undergone significant changes from POSIX.4a to POSIX.1c. These are listed in the following table:

Table 5-1: Thread Cancellation Control Interface Changes

POSIX.1c	POSIX.4a
`int pthread_setcancelstate (int state, int *oldstate)`	`int pthread_setcancel(int state)`
`int pthread_setcanceltype (int type, int *oldtype)`	`int pthread_setasynccancel (int state)`
`PTHREAD_CANCEL_ENABLE PTHREAD_CANCEL_ASYNCHRONOUS`	`CANCEL_ON`
`PTHREAD_CANCEL_DISABLE PTHREAD_CANCEL_DEFERRED`	`CANCEL_OFF`

The POSIX.4a interfaces use the values `CANCEL_ON` (to enable) and `CANCEL_OFF` (to disable) to control both the general cancellability state via **`pthread_setcancel()`** and the asynchronous cancellability state via **`pthread_setasynccancel()`**. POSIX.1c defines a distinct set of values for each control interface: `PTHREAD_CANCEL_ENABLE` and `PTHREAD_CANCEL_DISABLE` to control cancellability state via **`pthread_setcancelstate()`**, and `PTHREAD_CANCEL_DEFERRED` and `PTHREAD_CANCEL_ASYNCHRONOUS` to control cancellability type via **`pthread_setcanceltype()`**.

The **`pthread_cleanup_push()`** and **`pthread_cleanup_pop()`** functions in LynxOS versions prior to 3.1.0 returned an `int` instead of nothing (`void`) to

allow handing cleanup stack management errors from the underlying implementation. The LynxOS 4 versions of these functions have a `void` return type in compliance with POSIX.1c and silently ignore errors that may occur in the underlying implementation. This may result in one or more cleanup handlers not being registered or run if cleanup stack management errors occur.

Cancellability Defaults

In LynxOS versions prior to 3.1.0, both general and asynchronous cancellability were disabled by default in all newly created threads (this was not POSIX.4a Draft 4-compatible).

LynxOS 4 is fully compliant with POSIX.1c, where the cancellability state and type of any newly created threads, including the thread in which **`main()`** was invoked, is cancellation enabled (`PTHREAD_CANCEL_ENABLE`) and asynchronous cancellation disabled (`PTHREAD_CANCEL_DEFERRED`).

Cancellation Points

In POSIX.4a, cancellation points in blocking POSIX.1 and POSIX.4 (POSIX.1b) are implementation defined. LynxOS versions prior to 3.1.0 implemented cancellation points in the POSIX.1b blocking synchronization primitives, but not in POSIX.1 functions.

POSIX.1c explicitly defines function lists for all mandatory and optional POSIX.1 and C Standard cancellation points (see ISO/IEC 9945-1 1996 Section 18.1.2). LynxOS implements all listed optional as well as mandatory cancellation points.

Elimination of Cancellation-Safe Interfaces

POSIX.4a required a number of ANSI C Standard functions to be asynchronous-cancel safe. POSIX.1c does not require any POSIX.1 or C Standard functions to be asynchronous-cancel safe.

Cleanup Handler Restrictions

In POSIX.4a, support for exiting a cancellation handler via **`longjmp()`** or **`siglongjmp()`** was implementation defined, although not supported by LynxOS. This behavior is explicitly disallowed in POSIX.1c.

CHAPTER 6 *Synchronization Primitives*

Mutex Initialization Attributes

POSIX.4a defines no mutex attributes, whereas POSIX.1c defines process-shared and priority-scheduling attributes. The following table contains a comparison of the interfaces:

Table 6-1: Mutex Initialization Attributes Interface Changes

POSIX.1c	POSIX.4a
`int pthread_mutexattr_init (pthread_mutexattr_t *attr)`	`int pthread_mutexattr_create (pthread_mutexattr_t *attr)`
`int pthread_mutexattr_destroy (pthread_mutexattr_t *attr)`	`int pthread_mutexattr_delete (pthread_mutexattr_t *attr)`
`int pthread_mutexattr_setpshared (pthread_mutexattr_t *attr, int pshared)`	No equivalent
`int pthread_mutexattr_getpshared (const pthread_mutexattr_t *attr, int *pshared)`	No equivalent
`int pthread_mutexattr_setprotocol (pthread_mutex_attr *attr, int protocol)`	No equivalent

Table 6-1: Mutex Initialization Attributes Interface Changes (Continued)

POSIX.1c	POSIX.4a
`int pthread_mutexattr_getprotocol (const pthread_mutex_attr *attr, int *protocol)`	No equivalent
`int pthread_mutexattr_setprioceiling (pthread_mutexattr_t *attr,int prioceiling)`	No equivalent
`int pthread_mutexattr_getprioceiling (pthread_mutexattr_t *attr,int *prioceiling)`	No equivalent

In LynxOS, the POSIX.1c function **`pthread_mutexattr_destroy()`** sets the attribute object to an invalid value to prevent its subsequent use without reinitialization.

POSIX.1c defines the values `PTHREAD_PROCESS_SHARED` and `PTHREAD_PROCESS_PRIVATE` for the process-shared attribute.

NOTE: In LynxOS, the process-shared attribute can be set to either value but has no effect when the mutex is created, because the actual interprocess shareability of a mutex is determined by whether it is allocated in shared or process-private memory.

POSIX.1c defines the `PTHREAD_PRIO_NONE`, `PTHREAD_PRIO_INHERIT`, and `PTHREAD_PRIO_PROTECT` values for the `protocol` attribute. LynxOS supports all these values to provide normal, priority inheritance, and priority ceiling mutexes.

Mutex Initialization

POSIX.4a allows only dynamic mutex initialization. POSIX.1c supports static mutex initialization to default values using the `PTHREAD_MUTEX_INITIALIZER` macro, as follows:

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

This feature is useful in simplifying the initialization of libraries or other utility function packages that use mutexes by avoiding the need to use **`pthread_once()`** to ensure once-only initialization execution.

In POSIX.4a, the value **`pthread_mutexattr_default`** is used to dynamically initialize a mutex with default attributes. POSIX.1c specifies the use of `NULL` for this purpose.

Mutex Priority Ceiling Control

POSIX.1c provides interfaces for dynamically changing the priority ceiling of a mutex. Because POSIX.4a does not support mutex attributes, it has no corresponding interfaces, as shown in the following table:

Table 6-2: Mutex Priority Ceiling Control Interface Changes

POSIX.1c	POSIX.4a
`int pthread_mutex_setprioceiling (pthread_mutex_t *`*`mutex`*`,int *`*`prioceiling`*`,int *`*`old_ceiling`*`)`	No equivalent
`int pthread_mutex_getprioceiling (pthread_mutex_t *`*`mutex`*`,int *`*`prioceiling`*`)`	No equivalent

Condition Variable Initialization Attributes

POSIX.4a defines no condition variable attributes, whereas POSIX.1c defines process-shared attributes. The following table compares the corresponding interfaces:

Table 6-3: Condition Variable Initialization Attributes Interface Changes

POSIX.1c	POSIX.4a
`int pthread_condattr_init (pthread_condattr_t *attr)`	`int pthread_condattr_create (pthread_condattr_t *attr)`
`int pthread_condattr_destroy (pthread_condattr_t *attr)`	`int pthread_condattr_delete (pthread_condattr_t *attr)`
`int pthread_condattr_setpshared (pthread_condattr_t *attr, int pshared)`	No equivalent
`int pthread_condattr_getpshared (const pthread_condattr_t *attr, int *pshared)`	No equivalent

In LynxOS, the POSIX.1c function **`pthread_condattr_destroy()`** sets the attribute object to an invalid value to prevent its subsequent use without reinitialization.

POSIX.1c defines the values `PTHREAD_PROCESS_SHARED` and `PTHREAD_PROCESS_PRIVATE` for the process-shared attribute.

NOTE: In LynxOS, the process-shared attribute can be set to either value but has no effect when the condition variable is created, because the actual interprocess shareability of a condition variable is determined by whether it is allocated in shared or process-private memory.

Condition Variable Initialization

POSIX.4a allows only dynamic condition variable initialization. POSIX.1c supports static condition variable initialization to default values using the `PTHREAD_COND_INITIALIZER` macro, as follows:

```
pthread_cond_t condvar = PTHREAD_COND_INITIALIZER;
```

This feature is useful in simplifying the initialization of libraries or other utility function packages that use condition variables by avoiding the need to use **`pthread_once()`** to ensure once-only initialization execution.

In POSIX.4a, the value **`pthread_condattr_default`** is used to dynamically initialize a condition variable with default attributes. POSIX.1c specifies the use of `NULL` for this purpose.

CHAPTER 7 *Signals*

Controlling Blocked Signals

POSIX.4a defines the **`sigprocmask()`** function to examine and change blocked signals in both single-threaded and multi-threaded processes. In POSIX.1c, **`sigprocmask()`** must only be used in a single-threaded process; its behavior in a multi-threaded process is unspecified.

POSIX.1c defines the **`pthread_sigmask()`** function to examine and change blocked signals in either single-threaded or multi-threaded processes. Its behavior in a single-threaded process is the same as `sigprocmask()`.

Synchronously Accepting Signals

The **`sigwait()`** function is the mechanism by which a thread may wait for one or more asynchronous signals. The differences between the various versions of this function and the POSIX.1c version are shown in the following table:

Table 7-1: sigwait() Interface Changes

POSIX.1c	POSIX.4a	pre-LynxOS 3.1.0
`int sigwait (sigset_t *set,int *sig)`	`int sigwait (sigset_t *set)`	`int sigwait (sigset_t *set,void **value)`

LynxOS versions prior to 3.1.0 implemented a version of `sigwait()` based on the POSIX.4 Draft 9 event interface, which permits the return of an arbitrary value associated with the event causing the signal. POSIX.1c-compliant applications should instead use the **`sigwaitinfo()`** function for this purpose.

Thread-Based Event Notification

POSIX.1c defines a new mechanism for signal-based event notification: execution of a specified notification function as an independent thread. This mechanism is enabled by specifying the `SIGEV_THREAD` value for the **`sigev_notify`** member of the **`sigevent`** structure. Two new `sigevent` structure members specify the required thread information:

Table 7-2: New sigevent Structure Members

Member Name	Member Type
`sigev_notify_function`	`void (*)(union sigval)`
`sigev_notify_attributes`	`pthread_attr_t *`

The `sigev_notify_function` is executed in an environment as if it were the *`start_routine`* to **`pthread_create()`** with thread attributes specified by `sigev_notify_attributes`. As with `pthread_create()`, the value `NULL` can be used to specify default attributes. If the attributes are explicitly specified, the *`detachstate`* attribute must be set to `PTHREAD_CREATE_DETACHED`, otherwise the behavior is undefined.

The following code fragment demonstrates the usage of this capability to run a high-priority thread associated with the expiration of a high-resolution interval timer:

```
#include <pthread.h>
#include <signal.h>

void
timer_thread(union sigval value)
{
       int event_data;

       /* Retrieve the event data from the argument */
       event_data = value.sival_int;
       :
       /* Some high-priority processing here ... */
       :
       pthread_exit((void *)0);
}

int
main(int argc, char *argv[])
{
       int rv;
       int prio;
       pthread_attr_t attr;
       struct sched_param param;
       struct sigevent event;
       timer_t timer;
```

```
struct itimerspec timerspec;

:
/* Create and initialize the event thread */
/* attributes so that it will be run at a */
/* high, fixed priority */
rv = pthread_attr_init(&attr);
:
rv = pthread_setinheritsched(&attr,
PTHREAD_EXPLICIT_SCHED);
:
rv = pthread_setschedpolicy(&attr, SCHED_OTHER);
:
prio = sched_get_priority_max(SCHED_OTHER);
param.sched_priority = prio - 10;
rv = pthread_attr_setschedparam(&attr, &param);
:

rv = pthread_attr_setdetachstate(&attr,
PTHREAD_CREATE_DETACHED);
:
/* Initialize the event struct */
(void)memset(&event, 0, sizeof(event));
event.sigev_notify = SIGEV_THREAD;
event.sigev_notify_function = timer_thread;
event.sigev_value.sival_int = 0xdeadbeef;
event.sigev_notify_attributes = &attr;
:
/* Set up a 1 second periodic interval timer */
rv = timer_create(CLOCK_REALTIME, &event, &timer);
:
timerspec.it_interval.tv_sec = 1;
timerspec.it_interval.tv_nsec = 0;
timerspec.it_value.tv_sec = 1;
timerspec.it_value.tv_nsec = 0;
rv = timer_settime(timer, 0, &timerspec,
NULL);
:
/* Each time the interval timer expires,*/
/* the timer_thread will be run */
:
}
```

Signal Delivery Semantics

LynxOS versions prior to 3.1.0 could exhibit non-standard signal delivery behavior in a multi-threaded application where a signal was pending, but there were no threads blocked in `sigwait()` for that signal. In this case, the root thread would have the signal made pending against it, even if the it had the signal blocked but one or more of the other threads did not.

LynxOS 4 corrects the behavior for this case, ensuring that a pending signal is delivered to the first thread eligible to run that has the signal unblocked. Applications ported to LynxOS must not depend on the old behavior.

CHAPTER 8

Error Reporting

Pthread Function Errors

As noted in “Overview of Major Changes” on page 2, virtually all pthreads functions defined by POSIX.1c return the error number as the return value of the function instead of setting `errno` as in POSIX.4a, except where required by existing practice. This requires that error detection and handling for these functions be modified in a manner similar to the following code fragments:

POSIX.4a Code

```
:
if (pthread_create(&tid, &attr, thread1, NULL) == -1) {
perror(“pthread_create”);
exit(-1);
}
:
```

Equivalent POSIX.1c Code

```
:
int rv; // For function return value
:
rv = pthread_create(&tid, &attr, thread1, NULL);
if (rv != 0) {
fprintf(stderr,
       “ERROR: pthread_create returned %d\n”, rv);
exit(-1);
}
:
```

Because such POSIX.1c functions do not set `errno`, the application cannot use the C Standard function **`perror()`** to print the corresponding system error message. An alternative, but non-portable, method of printing the system error

message corresponding to the function return value on LynxOS 4 is to use the **ERRMSG** macro defined in **errno.h** to access the error message table:

```
rv = pthread_create(&tid, &attr, thread1, NULL);
if (rv != 0) {
fprintf(stderr,“pthread_create:%s\n”, ERRMSG(rv));
}
```

NOTE: Assigning the function return value to `errno` is *strongly* discouraged, especially to return status from non-root threads using `pthread_exit((void *)&errno)`.
Although `errno` is implemented as a per-thread variable in LynxOS, the storage for this per-thread data is persistent only during a thread's lifetime. After the thread terminates, this storage may be reused for other purposes at any time, rendering any error information invalid.

Elimination of EINTR Return Code

POSIX.4a allowed blocking pthreads functions to return an **EINTR** error if interrupted by a signal handler in a manner consistent with many blocking POSIX.1 or POSIX.1b interfaces. LynxOS versions prior to 3.1.0 exhibited this behavior, requiring the application to handle these situations.

POSIX.1c disallows this behavior, requiring all blocking pthreads functions to restart or complete after the execution of a signal handler transparently to the application. POSIX.1 or POSIX.1b blocking function behavior is unchanged.

APPENDIX A *Mapping Between Standards*

Mapping Standards

This appendix correlates sections in the POSIX.4a Draft 4 and POSIX.1 1996 (ISO/IEC 9945-1:1996) specifications. Since POSIX.1c is included in the POSIX.1 1996 standard, the sections and some of their contents are organized differently from those in POSIX.4a Draft 4.

Table A-1: POSIX 4a to POSIX.1 1996 Section Mapping

POSIX 4a Draft 4 Section/Topic	POSIX.1 1996 Section/Topic
2.7 Primitive System Data Types	2.5 Primitive System Data Types
2.9 C Language Definitions	2.7 C Language Definitions
2.11 Symbolic Constants	2.9 Symbolic Constants
3. Thread Management	16. Thread Management B.16 Thread Management Rationale
4. Thread Priority Scheduling	13.4 Thread Scheduling B.13 Execution Scheduling Rationale
4.3.3 Thread Yield	13.3.5 Yield Processor
5. Synchronization Primitives	11.3 Mutexes B.11.3 Mutex Rationale 11.4 Condition Variables B.11.4 Condition Variable Rationale
6. Thread Cancellation	18. Thread Cancellation B.18 Thread Cancellation Rationale
7. Thread-Specific Data	17. Thread-Specific Data B.17 Thread-Specific Data Rationale

Index

T

U