POSIX.1
Conformance Document

LynxOS Release 4

DOC-0414-00

Product names mentioned in *POSIX.1 Conformance* 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 Conformance Document* contains information about the conformance of LynxOS 4 to POSIX 1003.1: 1996.

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 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`

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`

POSIX.1 Conformance Document

Introduction

This document states the conformance of LynxOS 4 to POSIX 1003.1: 1996.

Throughout this document, the term POSIX.1 is used as an abbreviation for the POSIX 1003.1:1996 specification.

All heading numbers in this document correspond to heading numbers in the POSIX.1 specifications.

Certain areas of the FIPS 151-2 specification, which consists of the POSIX 1003.1 specification and a small set of modifications, allow implementation of additional options to meet specification requirements. However, the specification requires that all such optional behaviors be documented.

1. General

1.3 Conformance

1.3.1 Implementation Conformance

LynxOS systems must be configured as follows in order for an application to run with the behavior specified by POSIX.1.

No special actions or configurations are necessary to allow POSIX.1 applications to run with POSIX.1 semantics.

1.3.3 Language-Dependent Services for the C Programming Language

LynxOS meets the requirements of POSIX.1, Section 8 by reference to ISO/IEC 9989:1991, Information Technology-Programming Languages-C.

2. Terminology and General Requirements

2.2 Definitions

2.2.2 General Terms

The following terms are specific to the LynxOS implementation of POSIX.1. For details, see the LynxOS system documentation.

2.2.2.3 address space

These are the memory locations that can be accessed by the threads of a process.

2.2.2.4 appropriate privileges

Appropriate privileges are associated with a process by the following means: A process with an effective user ID of `0` has root privileges. A process without such privileges acquires them by calling **`setuid(0)`** or **`seteuid(0)`**. These calls only work from a `setuid` program with owner ID equal to `0`. An example of such a program is the **`su`** utility.

2.2.2.8 asynchronous I/O operation

Asynchronous I/O operation is an operation that does not cause the thread requesting the I/O to be blocked from further use of the processor.

This implies that the thread and the I/O operation may be running concurrently.

2.2.2.9 asynchronous I/O completion

Asynchronous read or write operations are complete when a corresponding status field is updated.

2.2.2.16 character special file

The following types of character special files are supported:

- Terminal device files
- Pseudo-terminal device files
- Raw SCSI device files
- Raw IDE device files
- Raw floppy device files
- Console device files
- Parallel port device files
- The kernel `syslog` device file
- The memory device file
- The null device file
- The controlling terminal file
- The NFS client device file
- The NFS server device file

2.2.2.14 blocked thread

A blocked thread is a thread that is waiting for some condition (other than the availability of a processor) to be satisfied before it can continue execution.

2.2.2.18 clock

A clock is an object that measures the passage of time.

The current value of time measured by a clock can be queried and possibly, set to a value within the legal range of the clock.

2.2.2.19 clock tick

A clock tick is an interval of time. A number of these occur each second. Clock ticks are among the units that may be used to express a value found in type **`clock_t`**.

2.2.2.20 condition variable

A synchronization object that allows a thread to suspend execution, repeatedly, until some associated predicate becomes true.

A thread whose execution is suspended on a condition variable is said to be *blocked* on the condition variable.

2.2.2.39 file

LynxOS supports the following file types:

- Regular files
- Character special files
- Block special files
- FIFO special files
- Directory files
- Contiguous files
- Sockets
- Symbolic links

2.2.2.62 map

To map is to create an association between a page-aligned range of the address space of a process and a range of physical memory or some memory object, such that a reference to an address in that range of the address space results in a reference to the associated physical memory or memory object.

The mapped memory or memory object is not necessarily memory resident.

2.2.2.63 memory object

A memory object is either a file or shared memory object.

When used in conjunction with **`mmap()`**, a memory object appears in the address space of the calling process.

2.2.2.65 message

A message is information that can be transferred among processes or threads by being added to and removed from a message queue. A message consists of a fixed size message buffer.

2.2.2.66 message queue

A message queue is an object to which messages can be added and removed.

Messages can be removed in the order they were sent or in priority order.

2.2.2.68 mutex

A mutex is a synchronization object used to allow multiple threads to serialize their access to shared data

This term is derived from the functionality it provides, namely, mutual exclusion. The thread that has locked a mutex becomes its owner and remains its owner until that same thread unlocks the mutex.

A thread that attempts to lock a mutex that is presently locked is blocked until the mutex is unlocked.

2.2.2.76 parent process ID

If a child process continues to exist after its creator process ceases to exist, a new parent process ID is assigned and the `init` process (process ID `1`) becomes the new parent.

2.2.2.78 path name

A path name that begins with two slashes is interpreted as being in the root directory. The double slash is resolved to a single slash.

2.2.2.80 persistence

This is a mode for semaphores,shared memory, and message queues requiring that the object and its state (including data, if any) are preserved after the object is no longer referenced by any process.

Persistence of an object does not imply that the state of the object is maintained across a system crash or system reboot.

2.2.2.84 preempted thread

This is a running thread whose execution is suspended due to another thread becoming runnable at a higher priority.

2.2.2.95 read-only file system

Modifications to objects on read-only file systems are restricted so that nothing on the file system can be modified, including access, update, and modification times in the inode.

2.2.2.98 reentrant function

This is a function whose effect, when called by two or more threads, is guaranteed to be as if each of the threads executed the function, one after the other in an undefined order, even if the actual execution is interleaved.

2.2.2.105 runnable thread

This is a thread that is capable of being a running thread, but for which no processor is available.

2.2.2.106 running thread

A running thread is one that is currently executing on a processor.

2.2.2.109 scheduling

Scheduling is the application of a policy to select a runnable thread to become a running thread, or to alter one or more thread lists.

2.2.2.111 scheduling contention scope

This is the property of a thread that defines the set of threads against which the given thread competes for resources.

For example, in a scheduling decision, threads sharing scheduling contention scope compete for processor resources.

In POSIX.1, a thread has a scheduling contention scope of either `PTHREAD_SCOPE_SYSTEM` or `PTHREAD_SCOPE_PROCESS`.

2.2.2.112 scheduling policy

This is the set of rules that is used to determine the order of thread execution.

In the context of POSIX.1, a scheduling policy affects thread ordering:

1. When a thread is running and it becomes a blocked thread.
2. When a thread is running and it becomes a preempted thread.

3. When a thread is a blocked thread and it becomes a runnable thread.
4. When a running thread calls a function that can change the priority or scheduling policy of a thread.

2.2.2.120 shared memory object

A shared memory object represents memory that can be mapped concurrently into the address space of more than one process.

2.2.2.121 signal

A signal is a mechanism by which a process may be notified of, or affected by, an event occurring in the system.

Examples of such events are hardware exceptions and specific actions by processes or threads. The term *signal* is also used to refer to the event itself.

2.2.2.123 supplementary group ID

A process's effective group ID is omitted from its list of supplementary group IDs.

2.2.2.130 synchronously generated signal

This is a signal that is attributable to a specific thread.

For example, a thread executing an illegal instruction or touching invalid memory causes a synchronously generated signal. Synchronicity is a property of how the signal was generated, and not a property of the signal number.

2.2.2.131 system

This is an implementation of this part of ISO/IEC 9945.

2.2.2.132 system crash

This is an interval initiated by an unspecified circumstance that causes all processes (possibly other than special system processes) to be terminated in an undefined manner. After a crash, any changes to the state and contents of files written to by a conforming POSIX.1 application prior to the interval are undefined, except as required elsewhere in POSIX.1.

2.2.2.133 system process

This is an object, other than a process executing an application, that is defined by the system and has a process ID.

2.2.2.134 system reboot

This is an implementation defined sequence of events that may result in the loss of transitory data, i.e. data that is not saved in permanent storage.

This includes message queues, shared memory, semaphores, and processes.

2.2.2.136 thread

A thread is a single flow of control within a process.

Each thread has its own thread ID, scheduling priority and policy, `errno` value, thread-specific key/value bindings, and the required system resources to support a flow of control. Anything whose address may be determined by a thread, including but not limited to static variables, storage obtained by **`malloc()`**, directly addressable storage obtained through implementation supplied functions, and automatic variables are accessible to all threads in the same process.

2.2.2.137 thread ID

This is a unique value of type **`pthread_t`** that identifies each thread during its lifetime in a process.

2.2.2.138 thread list

This is an ordered set of runnable threads that all have the same ordinal value for their priority.

The ordering of threads on the list is determined by a scheduling policy or policies. The set of thread lists includes all runnable threads in the system.

2.2.2.139 thread-safe

This defines a function that may be safely invoked concurrently by multiple threads.

Each function in POSIX.1 is thread-safe unless explicitly stated otherwise. An example is any “pure” function (a function that holds a mutex locked while it is accessing static storage or objects shared among threads).

2.2.2.140 thread-specific data key

This is a process global handle of type **`pthread_key_t`** that is used for naming thread-specific data.

Although the key value may be used by different threads, the values bound to the key by **`pthread_setspecific()`** and accessed by **`pthread_getspecific()`** are maintained on a per-thread basis and persist for the life of the calling thread.

2.2.2.141 timer

This is an object that can notify a process when the time measured by a particular clock has reached or passed a specified value, or when a specified amount of time, as measured by a particular clock, has elapsed.

2.2.2.142 timer overrun

This is a condition that occurs each time a timer, for which there already is an expiration signal queued to the process, expires.

2.3 General Concepts

2.3.1 extended security controls

The LynxOS operating system does not implement any extended security controls.

2.3.2 file access permissions

LynxOS does not provide any additional or alternative file access control mechanisms.

2.4 Error Numbers

In addition to the errors listed in this clause, LynxOS supports the following errors under the stated conditions:

[ETXTBSY]	Text file is busy. An attempt was made to write to (or request write accessibility of) a file that is currently being executed; or to execute a file that is currently being written.

The interfaces that may set off this error are documented in the sections of this document corresponding to the descriptions of those interfaces in POSIX.1.

On LynxOS systems, the [EFBIG] error does not occur because there is no maximum file size.

2.5 Primitive System Data Types

In addition to the primitive system data types listed in Table 2-1 in the standard, the following types, whose names end with **_t**, are defined in headers specified by POSIX.1.

Defined Type	Header	Description
__st_recovery_t	stdio.h (via st.h)	Thread cancellation
caddr_t	sys/types.h	C code address type
cc_t	termios.h	Type for termios.c_cc
clock_t	sys/times.h & time.h	ANSI C time type
clockid_t	sys/types.h	POSIX clock ID type
csem_t	stdio.h (via sem.h)	Counting semaphore
cv_t	stdio.h (via sem.h)	Condition variable
div_t	stdlib.h	ANSI C type for div()
fpos_t	stdio.h	ANSI C file position
int16_t	sys/types.h	16-bit signed integer
int32_t	sys/types.h	32-bit signed integer
int64_t	sys/types.h	64-bit signed integer
int8_t	sys/types.h	8-bit signed integer
key_t	sys/types.h	Type for ftok()
ldiv_t	stdlib.h	ANSI C type for ldiv()
mutex_t	stdio.h (via sem.h)	Mutex
ptrdiff_t	stddef.h	ANSI C pointer difference
sig_atomic_t	signal.h	ANSI C atomic type
sigaction_t	signal.h	Arguments to sigaction()

Defined Type	Header	Description
siginfo_t	signal.h (via signal.p4.h)	Signal information
sigset_t	signal.h	Signal set
sigwhdr_t	signal.h	sigwaiter linked list
sigwptr_t	signal.h	sigwaiter
sigwvec_t	signal.h	sigwhdr_t for 64 sigs
speed_t	termios.h	Terminal baud rate
st_attr_t	stdio.h (via st.h)	Thread attributes
synch_t	stdio.h (via sem.h)	Synchronization object
tcflag_t	termios.h	struct termios flags
threadspec_t	stdio.h (via st.h)	Thread-specific data
tid_t	sys/types.h	Thread ID
time_t	sys/types.h & time.h	ANSI C time type
timer_t	sys/types.h	POSIX timer ID type
u_int16_t	sys/types.h	16-bit unsigned int
u_int32_t	sys/types.h	32-bit unsigned int
u_int64_t	sys/types.h	64-bit unsigned int
u_int8_t	sys/types.h	8-bit unsigned int
ulong_t	sys/types.h	Unsigned long
ushort_t	sys/types.h	Unsigned short
usynch_chk_t	stdio.h (via sem.h)	
wchar_t	stddef.h & stdlib.h	ANSI C wide character

2.6 Environment Description

LynxOS permits the following characters, in addition to the portable file name character set, in environment variable names:

```
!@#$%^&*()+{}[]:;'"<>,/?`~\|
```

2.7 C Language Definition

2.7.2 POSIX.1 Symbols

The following additional feature test macro is defined for LynxOS:

```
_POSIX_SOURCE
```

2.8 Numerical Limits

2.8.3 Run-Time Increasable Values

{NGROUPS_MAX} is set to 8 in <limits.h>.

2.8.4 Run-Time Invariant Values (Possibly Indeterminate)

The following magnitude limitations are contained in <limits.h>:

Name	Value	Comments
ARG_MAX	65536	Depends on EXECARGLEN in <conf.h>
TZNAME_MAX	10	

2.8.5 Path Name Variable Values

The values in the following table are constant from one path name to another:

Name	Value	Comments
LINK_MAX	32767	Depends on LMAX_LINKS in <conf.h>
MAX_CANON	256	Depends on LINELEN in <ttymgr.h>
MAX_INPUT	512	Depends on LINELEN in <ttymgr.h>
NAME_MAX	255	Depends on MAXNAMLEN in <conf.h>
PATH_MAX	1024	Depends on MAXPATHLEN in <conf.h>
PIPE_BUF	512	

2.9 Symbolic Constants

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

<unistd.h> contains the following values:

Symbolic Constant	Value	Comments
_POSIX_JOB_CONTROL	1	
_POSIX_SAVED_IDS	1	

2.9.4 Execution-Time Symbolic Constants for Portability Specifications

<unistd.h> contains the following values:

Symbolic Constant	Value	Comments
_POSIX_CHOWN_RESTRICTED	1	Applies to all files
_POSIX_NO_TRUNC	1	Applies to all files
_POSIX_VDISABLE	0	Applies to all terminals

3. Process Primitives

3.1 Process Creation and Execution

3.1.1 Process Creation

3.1.1.2 Description

Each open directory stream in the child process shares directory stream positioning with the corresponding directory stream of the parent.

All inherited and non-inherited process characteristics are documented under **fork** in Section 2 of the online man pages.

3.1.1.4 Errors

If the **fork()** function fails, a value of -1 is returned to the parent process. No child is created and an errno indicates the error:

- [EAGAIN]=The maximum number of processes that can be running on the system (NPROC) is reached, or no thread enries are available to create a new process.
- [ENOMEM]=Not enough memory on the system to copy the parent's memory.

3.1.2 Execute a File

3.1.2.2 Description

When constructing a path name that identifies a new process image file, if the file argument does not contain a slash and the **PATH** environment variable is not present, the results of a search for the file are defined as follows: Only the current directory is searched.

All inherited and non-inherited process characteristics are documented under **execve** in Section 2 of the online man pages.

3.1.2.4 Errors

LynxOS supports the execution of only regular files.

For the **exec** type functions, LynxOS detects the conditions and returns the corresponding errno value for [ENOMEM].

- On LynxOS systems, the exec type functions return -1 and set errno to [ETXTBSY] if the file is already open for writing.

3.2 Process Termination

3.2.1 Wait for Process Termination

3.2.1.2 Description

The **wait()** and **waitpid()** functions may report the status of any traced child that is in the stopped state. A child process is traced if it has called **ptrace()** with

a first argument of `PTRACE_TRACME`, or if its parent process has called `ptrace()` with a first argument of `PTRACE_ATTACH` and a second argument equal to the child's process ID.

3.2.2 Terminate a Process

3.2.2.2 Description

Children of a process terminated by **`_exit()`** are assigned the `init` process (process ID `1`) as their parent process.

3.3 Signals

3.3.1 Signal Concepts

3.3.1.1 Signal Names

The following additional signals beyond those required by POSIX.1 occur in LynxOS:

Signal Name	Description
`SIGTRAP`	Trace trap - debugger trap
`SIGCORE`	Kill with core dump (sent by the user)
`SIGSYS`	Bad system call number
`SIGURG`	Urgent condition (data) on socket
`SIGIO`	I/O possible on descriptor (sent when I/O is possible (data has arrived) on a file descriptor on which an `fcntl(..., FASYNC)` was performed.)
`SIGVTALRM`	Virtual time alarm-This signal is sent when a virtual timer (set by `setitimer (ITIMER_VIRTUAL)`) expires.
`SIGPROF`	Profiling alarm-This signal is sent when a profile timer (set by `setitimer(ITIMER_PROFILE)`) expires.
`SIGWINCH`	Window size change
`SIGPRIO`	Sent to a process when its priority or process group is changed

3.3.1.2 Signal Generation and Delivery

If a subsequent occurrence of a pending signal is generated, the signal is not delivered more than once.

Signals are generated under the following conditions not specified in POSIX.1:

Signal Name	Generation Condition
SIGTRAP	Trace trap - debugger trap
SIGCORE	Kill with core dump (sent by the user)
SIGBUS	Bus error (user program access non-existent, or non-aligned memory
SIGSYS	Bad system call number
SIGURG	Urgent condition (data) on socket
SIGIO	I/O possible on descriptor (sent when I/O is possible (data has arrived) on a file descriptor on which an `fcntl(..., FASYNC)` was performed.)
SIGVTALRM	Virtual time alarm-This signal is sent when a virtual timer (set by `setitimer(ITIMER_VIRTUAL)`) expires.
SIGPROF	Profiling alarm-This signal is sent when a profile timer (set by `setitimer(ITIMER_PROFILE)`) expires.
SIGWINCH	Window size change
SIGPRIO	Sent to a process when its priority or process group is changed

3.3.2 Send a Signal to a Process

3.3.2.2 Description

On LynxOS systems, **`kill(pid, sig)`** exhibits the following implementation-specific behavior:

> If `pid` is zero, `sig` is sent to all processes whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal, excluding the null process (process ID `0`) and the `init` process (process ID 1).

LynxOS does not provide extended security controls and does not impose any restrictions on the sending of signals, including the null signal.

3.3.3 Manipulate Signal Sets

3.3.3.4 Errors

For **`sigaddset()`**, **`sigdelset()`**, and **`sigismember()`**, LynxOS detects the conditions and returns the corresponding `errno`.

- `[EINVAL]`=The argument is not a valid signal number

3.3.6 Examine Pending Signals

3.3.6.4 Errors

On LynxOS systems, the following error condition is detected for **`sigpending()`**:

- `[EFAULT]`=The argument does not point to allocated memory in the process' address space.

4. Process Environment

4.2. User Identification

4.2.2 Set User and Group IDs

4.2.2.2 Description

These are the ways in which a process obtains appropriate privileges are described in Section 2.2.2.4 of this document.

4.4 System Identification

4.4.1 Get System Name

4.4.1.2 Description

The **utsname** structure members and corresponding formats are listed in the following table.

Member Name	Format
sysname	char[256]
nodenam	char[256]
release	char[9]
version	char[9]
machine	char[9]

4.4.1.4 Errors

On LynxOS systems, the following error condition is detected for **uname()**:

- [EFAULT]=The argument does not point to allocated memory in the process' address space.

4.5 Time

4.5.1 Get System Time

4.5.1.4 Errors

On LynxOS systems, no error conditions are detected for **time()**.

4.5.2 Get Process Times

4.5.2.4 Errors

The following error condition is detected for **`times()`**:

[EFAULT] The argument does not point to allocated memory in the process' address space.

4.6 Environment Variables

4.6.1 Environment Access

4.6.1.4 Errors

No error conditions are detected for **`getenv()`**.

4.7 Terminal Identification

4.7.1 General Terminal Path Name

4.7.1.4 Errors

No error conditions are detected for **`ctermid()`**.

4.7.2 Determine Terminal Device Name

4.7.2.4 Errors

No error conditions are detected for **`ttyname()`** or **`isatty()`**.

4.8 Configurable System Variables

4.8.1 Get Configurable System Variables

4.8.1.2 Description

LynxOS does not support any system variables for **`sysconf()`** beyond those listed in Table 4-2 in the standard.

5. Files and Directories

5.1 Directories

5.1.1 Format of Directory Entries

A directory's link count is incremented when a subdirectory is created.

5.1.2 Directory Operations

5.1.2.2 Description

If the **`dirp`** argument does not point to an open directory stream, **`readdir()`** returns a `NULL` pointer and sets `errno` to `[EBADF]`.

For **`opendir()`**, LynxOS detects the conditions and returns the corresponding `errno` values for `[EMFILE]` and `[ENFILE]`.

For **`closedir()`**, LynxOS detects the conditions and returns the corresponding `errno` value for `[EBADF]`.

5.2 Working Directory

5.2.2 Get Working Directory Path Name

5.2.2.4 Errors

For **`getcwd()`**, LynxOS detects the conditions and returns the corresponding `errno` value:

- `[EINVAL]`=size is less than or equal to zero.
- `[ERANGE]`=size is greater than zero but smaller than the length of the pathname plus one.
- `[EACCESS]`=A directory in the path denies read or search permission.

5.3 General File Creation

5.3.1 Open a File

5.3.1.2 Description

A newly-created file's group ID is set to the group ID of the parent directory.

If **`O_TRUNC`** is set in `oflag` and `PATH` refers to a file type other than a regular file, a FIFO special file, or a terminal device file, the effect of **`open(path, oflag, mode)`** is as follows:

For contiguous files, the file size is set to `0` (`ls -l` shows a `0` size for the file), but its block size (the number of blocks set aside for the contiguous file) is unchanged. For all other file types, the `O_TRUNC` flag is ignored.

5.3.1.4 Errors

On LynxOS systems, the **`open()`** function returns `-1` and sets `errno.`

- `[EACCES]`=Requested access for file denied according to the mode of the file, or a directory in the path denied search permission.
- `[EEXIST]`=`O_EXCL` was requested, and the file exists.
- `[EFAULT]`=path does not point to allocated memory in the process's address space.

- [EINTR]=A signal interrupted the open call.
- [EISDIR]=The file is a directory, and write access was requested.
- [ELOOP]=Too many symbolic links were encountered when traversing the path.
- [EMFILE]=The maximum number of files open per process has been reached.

5.3.3 Set File Creation Mask

5.3.3.2 Description

When bits other than the file permission bits are set in the **mask** argument to **umask()**, all bits other than the file permission bits are ignored.

5.3.3.3 Errors

No error conditions are detected by mask().

5.3.4 Link to a File

5.3.4.2 Description

link(existing,new) does not succeed when existing and new refer to locations on different file systems.

link() is not supported on directories.

The calling process need not have permission to access the existing file when linking files.

5.3.4.3 Errors

Link returns 0 if the link was made, or -1 if not. If the call fails, errno will contain one of the following:

- [ENOTDIR] =An element of either path prefix is not a directory.
- [ENAMETOOLONG] =path1 or path2 is too long.
- [ENOENT] =An element of either path prefix does not exist.
- [ENOENT] =path1 does not exist.

- [EEXIST] =path2 does exist.
- [EPERM] =path1 refers to a directory.
- [ELOOP] =Too many symbolic links were encountered when traversing one of the paths.
- [EROFS] =Making the link requires writing to a directory on a read-only file system.
- [EACCES] =Making the link requires writing to a directory that denies write permission.
- [EFAULT]=path1 or path2 does not point to allocated memory in the process' address space.
- [EXDEV]=path1 and path2 are on different file systems.

5.4 Special File Creation

5.4.1 Make a Directory

5.4.1.2 Description

When bits other than the file permission bits are set in the `mode` argument to **`mkdir()`**, the `mkdir()` function ignores them.

A newly-created directory's group ID is set to the group ID of the parent directory.

5.4.1.3 Errors

`mkdir` returns 0 if the directory was created, or -1 if not. If the call fails, `errno` will contain one of the following:

- [ENOTDIR]=An element of the path prefix is not a directory.
- [ENAMETOOLONG]=The path name is too long.
- [ENOENT]= An element of the path prefix does not exist.
- [EEXIST]=The named file already exists.
- [ELOOP]=Too many symbolic links were encountered when traversing the path.
- [EACCES]= A directory in the path denied search permission, or the parent directory of the directory to be created denied write permission.

- [EFAULT]=path does not point to allocated memory in the process' address space.
- [EIO]= An I/O error occurred.

5.4.2 Make a FIFO Special File

5.4.2.2 Description

When bits other than the file permission bits are set in the mode argument to **mkfifo()**, the mkfifo() function ignores them.

A newly-created FIFO's group ID is set to the group ID of the parent directory.

5.4.2.3 Errors

If any of the following conditions occur, the mkfifo() function shall return -1 and set errno to the corresponding value;

- [EACCES]= Search permission is denied for a component of the path prefix or write permission is denied on the parent directory of the file to be created
- [EEXIST]= The named file already exists.
- [ENAMETOOLONG]= The length of the path string exceeds PATH_MAX, or a pathname component is longer than {NAME_MAX} while _POSIX_NO_TRUNC is in effect.
- [ENOENT]= A component of the path prefix does not exist, or the path argument points to an empty string.
- [ENOSPC]= The directory that would contain the new file cannot be extended, or the file system is out of file allocation resources.
- [ENOTDIR]= A component of the path prefix is not a directory.
- [EROFS]=The named file resides on a read-only file system.

5.5 File Removal

5.5.1 Remove Directory Entries

LynxOS does not support the use of **unlink()** on directories.

5.5.2 Remove a Directory

5.5.2.2 Description

If the named directory is the root directory or the current working directory of any process, **`rmdir()`** fails and sets `errno` to `[EBUSY]`.

5.5.2.4 Errors

If the directory indicated in the call to `rmdir()` is being used by another process, `rmdir()` succeeds.

`rmdir` returns 0 if successful, or -1 if not. If the call fails, errno will contain one of the following:

- `[ENOTDIR]`= An element of the path is not a directory.
- `[ENAMETOOLONG]`= The path name is too long.
- `[ENOENT]`= The directory does not exist.
- `[ELOOP]`= Too many symbolic links were encountered when traversing the path.
- `[EACCES]`= Write access for directory denied according to the mode of the file, or a directory in the path denied search permission.
- `[EFAULT]`= path does not point to allocated memory in the process' address space.
- `[EBUSY]`= The directory is the root of a mounted system.
- `[EROFS]`= The directory is on a read-only file system.
- `[ENOTEMPTY]`= The directory has entries other than just “.” and “..”

5.5.3 Rename a File

5.5.3.2 Description

In a call to **`rename(old, new)`**, if the `old` argument points to the path name of a directory, write access permission is required for the directory named by `old`, and, if it exists, for the directory named by `new`.

5.5.3.4 Errors

In a call to `rename(path1, path2)`, if the `old` argument and, if it exists, the `new` argument points to the path name of a directory and one of the directories indicated in the call to `rename()` is in use by the system or by another process, `rename()` fails.

5.6 File Characteristics

5.6.2 Get File Status

5.6.2.2 Description

LynxOS provides no additional or alternate file access control mechanisms that cause **`stat()`** or **`fstat()`** to fail. Information included in the stat struct is defined in `/usr/include/sys/stat.h`:

```
struct stat {
            dev_t   st_dev;     /* block device inode is on */
            ino_t   st_ino;     /* inode number */
            int     st_mode;    /* protection and file type */
            int     st_nlink;   /* hard link count */
            int     st_uid;     /* user id */
            int     st_gid;     /* group id */
            dev_t   st_rdev;    /* the device number for a special file */
            off_t   st_size;    /* number of bytes in a file */
            time_t  st_atime;   /* time of last access */
            time_t  st_mtime;   /* time of last modify */
            time_t  st_ctime;   /* time of last status change */
            long    st_blksize; /* size of a block */
            long    st_blocks;  /* number of blocks in the file */
          };
```

5.6.3 Check File Accessibility

5.6.3.4 Errors

For **`access()`**, LynxOS detects the conditions and returns the corresponding `errno` value for `[EINVAL]`.

On LynxOS systems, the `access()` function returns `-1` and sets `errno` to `[ETXTBSY]` if write permission (`W_OK`) is being requested, and the file is currently being executed.

If the call fails `errno` will contain one of the following:

- `[ENOTDIR]`= An element of the path prefix is not a directory.

- [ENOENT]= The path name is too long, or the file does not exist.
- [EPERM]= The path name contains a non-ASCII character.
- [ELOOP]= Too many symbolic links were encountered when traversing the path.
- [EROFS]= Write access is denied because file is on a read-only file system.
- [ETXTBSY]= Write access is denied because the file is being executed.
- [EACCES]= Requested access for file denied according to the mode of the file, or a directory in the path denied search permission.
- [EFAULT]= path does not point to allocated memory in the process' address space.

5.6.4 Change File Modes

5.6.4.2 Description

chmod() has the following effect on file descriptors that refer to files that are open at the time of the call:

The chmod() call takes effect immediately, but does not affect the availability of data from a file descriptor or stream that is already open.

5.6.4.3 Errors

chmod returns 0 if successful, or -1 if not. If chmod fails, errno will contain one of the following:

- [ENOTDIR]=An element of the path prefix is not a directory.
- [ENAMETOOLONG]=The path name is too long.
- [ENOENT]=The file does not exist.
- [ELOOP]= Too many symbolic links were encountered when traversing the path.
- [EACCES]=A directory in the path denied search permission.
- [EFAULT]=path does not point to allocated memory in the process's address space.
- [EROFS]=The file is on a read-only file system.

5.6.5 Change Owner and Group of a File

5.6.5.2 Description

If the `path` argument to **`chown()`** refers to a regular file and if the call is made by a process with appropriate privileges, the set-user-ID (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode are cleared upon successful return from `chown()`.

5.6.5.4 Errors

For `chown()`, LynxOS does not detect the conditions and does not return the corresponding `errno` value for `[EINVAL]`.

`chown` and `fchown` return 0 if successful, -1 if not. If an error occurs `errno` will contain one of the following values:

- `[EPERM]`= The effective user ID of the calling process is not 0, and either the process's effective user ID is not equal to the file's user ID, uid is not equal to the file's user ID, or gid is not equal to the process's group ID or one of its supplementary groups IDs.
- `[ENOTDIR]`= An element of the path prefix is not a directory.
- `[ENAMETOOLONG]`= The path name is too long.
- `[ENOENT]`= The file does not exist.
- `[ELOOP]`= Too many symbolic links were encountered when traversing the path.
- `[EACCES]`= A directory in the path denied search permission.
- `[EFAULT]`=path does not point to allocated memory in the process' address space.
- `[EROFS]`= The file is on a read-only file system. If fchown fails, errno will contain one of the following:
- `[EBADF]`= fd is not a valid file descriptor.
- `[EINVAL]`= fd refers to a pipe.
- `[EROFS]`= The file is on a read-only file system.

If `fchown` fails, `errno` will contain one of the following:

- `[EBADF]`=fd is not a valid file descriptor.

- [EINVAL]=fd refers to a pipe.
- [EROFS]=The file is on a read-only file system.

5.7 Configurable Path Name Variables

5.7.1 Get Configurable Path Name Variables

5.7.1.2 Description

LynxOS does not support any variables for **pathconf()** beyond those listed in Table 5-2 in the standard.

5.7.1.4 Errors

For pathconf(), LynxOS detects the conditions and returns the corresponding errno values for the following:

- [EINVAL]
- [EACCES]
- [ENAMETOOLONG]
- [ENOENT]
- [ENOTDIR]

For **fpathconf()**, LynxOS detects the conditions and returns the corresponding errno values for the following:

- [EBADF]
- [EINVAL]

6. Input and Output Primitives

6.4 Input and Output

6.4.1 Read from a File

6.4.1.2 Description

If **`read()`** is interrupted by a signal after it has successfully read some data, it returns the number of bytes read.

If the `fildes` argument refers to a device special file, `read()` requests made after an end-of-file indication has been returned have the following result: Nothing is read, no error occurs, and a count of zero bytes is returned.

If the value of the `count` argument to `read()` is greater than `{SSIZE_MAX}`, no error occurs, and zero is returned.

6.4.2 Write to a File

6.4.2.2 Description

If **`write()`** is interrupted by a signal after it successfully writes some data, it returns the number of bytes written.

If the value of `nbyte` is greater than `{SSIZE_MAX}`, the result of a call to `write(fd, buff, count)` is as follows: No error occurs, and zero is returned.

6.5 Control Operations on Files

6.5.2 File Control

6.5.2.2 Description

Advisory record locking is supported for the following types of files:

- Regular files
- Block special files
- Contiguous files

6.5.2.4 Errors

If **`fcntl()`** fails, `errno` will contain one of the following:

- `[EBADF]`= fd is not a valid open file descriptor.
- `[EBADF]`=cmd is F_SETLK or F_SETLKW and the process does not have the appropriate read or write permissions on the file.
- `[EMFILE]`=There are no more free file descriptors.
- `[EINVAL]`=cmd is F_DUPFD and arg is not a valid file descriptor number.
- `[EINVAL]`=cmd is F_GETLK, F_SETLK, or F_SETLKW and the flock structure pointed to by arg is not valid.
- `[EINVAL]`=cmd is not one of the valid commands.
- `[EFAULT]`= cmd is F_GETLK, F_SETLK, or F_SETLKW and arg does not point to allocated memory in the process address space.
- `[EACCES]`= cmd is F_SETLK, the lock type (l_type) is F_RDLCK (shared lock), and the segment of the file to be locked already has an exclusive lock held by another process, or the lock type is F_WRLCK (exclusive lock) and the segment of the file to be locked already has either a shared or exclusive lock held by another process.
- `[ENOLCK]`= cmd is F_SETLK or F_SETLKW, a lock is being created (F_RDLCK or F_WRLCK), and there are no more file lock entries available.

- [EDEADLK]=cmd is F_SETLKW, the lock is blocked by another process' lock, and suspending the calling process would cause a deadlock situation to occur.
- [EDEADLK]=cmd is F_SETLK or F_SETLKW, a lock is being removed (F_UNLCK), and there are no more file lock entries available.

6.5.3 Reposition Read/Write File Offset

6.5.3.2 Description

lseek() behaves as follows on devices incapable of seeking:

- For pipes, fifos, and UNIX domain sockets, lseek() returns -1 and sets errno to [ESPIPE].
- For non-UNIX domain sockets, lseek() returns zero.

6.7 Asynchronous I/O

There are no circumstances under which the priority ordering of asynchronous I/O requests is relaxed.

The relative priority of asynchronous I/O versus synchronous I/O is identical.

All asynchronous I/O operations are cancellable.

7. Device- and Class-Specific Functions

7.1 General Terminal Interface

LynxOS supports asynchronous ports, network connections, and synchronous ports using the General Terminal Interface defined in POSIX.1.

7.1.1 Interface Characteristics

7.1.1.3 The Controlling Terminal

If a session leader has no controlling terminal and opens a terminal device that is not already associated with a session with `O_NOCTTY` clear, the terminal becomes the controlling terminal of the session leader.

7.1.1.5 Input Processing and Reading Data

If a limit exists on the number of bytes that may be stored in the input queue, and this limit is exceeded, additional bytes are discarded.

7.1.1.6 Canonical Mode Input Processing

When there are `{MAX_CANON}` characters in the input queue for a given terminal device for which **`MAX_CANON`** is supported, each subsequent character is ignored and the audible bell character (**Ctrl-G**) is echoed.

7.1.1.8 Writing Data and Output Processing

LynxOS provides a buffering mechanism for `write()`s to a terminal device. The written bytes are added to a queue. A separate task, called a kernel thread, reads the bytes from this queue and actually performs the transmission to the terminal device.

7.1.1.9 Special Characters

The `START` and `STOP` characters may be changed.

7.1.1.10 Modem Disconnect

After a modem disconnect occurs for a controlling terminal (where **`CLOCAL`** is not set in the **`c_cflag`** field for the terminal), if a process in a background process group attempts to read from its controlling terminal, and either the process is ignoring or blocking the `SIGTTIN` signal or the process group of the process is orphaned, the `read()` call returns `-1` and sets `errno` to `[EIO]`.

7.1.2 Parameters That Can Be Set

7.1.2.2 Input Modes

A break condition is not defined in any context other than asynchronous serial data transmission.

STOP and START characters are transmitted under the following conditions:

- When **`tcflow()`** is called with action argument of `TCIOFF`, then `STOP` is transmitted
- When **`ioctl()`** is called with a request value of `TIOCIOFF`, then `STOP` is transmitted
- When a hangup condition is detected and `CLOCAL` is not set in `c_cflag`, then a `STOP` followed by a `START` is transmitted
- When **`tcflow()`** is called with action equal to **`TCION`**, then `START` is transmitted
- When `ioctl()` is called with request of **`TIOCION`**, then `START` is transmitted

The initial input control value after `open()` is:

- `BRKINT` - set
- `ICRNL` - set
- `IGNBRK` - not set
- `IGNCR` - not set
- `IGNPAR` - set
- `INLCR` - not set
- `INPCK` - not set
- `ISTRIP` - not set
- `IXOFF` - not set
- `IXON` - set
- `PARMRK` - not set

7.1.2.3 Output Modes

Terminal output processing when `OPOST` is set in `c_oflag` is as follows:

- If **`ONLCR`** is set in `c_oflag`, then `<newline>` is mapped to `<carriage_return><newline>` on output. Also, upon line termination, `<carriage_return><newline>` is echoed.
- If **`ONLRET`** is set in `c_oflag`, then `<newline>` is echoed as `<carriage_return>`.
- If the **`TABDLY`** bits in `c_oflag` are set to `TAB3`, then on output, tab characters are replaced with the appropriate number of space characters such that the tabs occur every 8 characters. Also, tabs are echoed in the same fashion.

The initial output control value after `open()` is as follows:

- `OPOST` - set
- `ONLCR` - set

7.1.2.4 Control Modes

The initial hardware control values after `open()` are as follows:

- `CLOCAL` - not set
- `CREAD` - set
- `CSIZE` - `CS8`
- `CSTOPB` - not set
- `HUPCL` - not set
- `PARENB` - not set
- `PARODD` - not set

7.1.2.5 Local Modes

If **`IEXTEN`** is set, the following functions are recognized from the input data:

- Recognize reprint character
- Recognize word erase character
- Recognize BSD `dsuspc` character

- Recognize literal character
- Allow the feature of restarting suspended input on any character (DECCTLQ) to be enabled/disabled

When IEXTEN has been set it interacts with **ICANON**, **ISIG**, **IXON**, or **IXOFF** in the following manner:

If ICANON is set (independent of ISIG), the following are enabled:

- Recognize reprint character
- Recognize word erase character
- Recognize BSD dsuspc character
- Echo control character as ^<char>

If ISIG is set and ICANON is not set:

- Echo control character as ^<char> is enabled

IXON and IXOFF do not affect IEXTEN

Regardless of the values of ICANON and ISIG, the ability to enable or disable the "restart suspended input on any char (DECCTLQ)" facility is possible using the IXANY bit of c_iflag.

The initial local control value after open() is:

- ECHO - set
- ECHOE - set
- ECHOK - set
- ECHONL - not set
- ICANON - set
- IEXTEN - set
- ISIG - set
- NOFLSH - not set
- TOSTOP - not set

7.1.2.6 Special Control Characters

The initial values of control characters are defined as follows:

Special Control Characters		
Canonical Mode	Non-Canonical Mode	Initial Value
VEOF		4 (**Ctrl-D**)
VEOL		-1
VERASE		8 (**Ctrl-H**)
VINTR	VINTR	3 (**Ctrl-C**)
VKILL		21 (**Ctrl-U**)
	VMIN	1
VQUIT	VQUIT	28 (**Ctrl-**)
VSUSP	VSUSP	26 (**Ctrl-Z**)
	VTIME	0
VSTART	VSTART	17 (**Ctrl-Q**)
VSTOP	VSTOP	19 (**Ctrl-S**)

7.1.3 Baud Rate Functions

7.1.3.2 Description

If an attempt is made to set an unsupported baud rate, **`cfsetospeed()`** and **`cfsetispeed()`** perform no actions.

7.1.3.4 Errors

No error conditions are detected for the following:

- `cfgetospeed()`
- `cfsetospeed()`
- `cfgetispeed()`
- `cfsetispeed()`

7.2 General Terminal Interface Control Functions

7.2.1 Get and Set State

LynxOS supports using **`tcsetattr()`** to set a terminal's input and output baud rates to different values.

7.2.2 Line Control Functions

7.2.2.2 Description

If the duration parameter to **`tcsendbreak()`** is not zero, zero-valued bits are sent for 0.25 seconds.

7.2.2.3 Errors

Upon successful completion, a value of zero is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error. If any of the following conditions occur, the `tcsendbreak()` function returns -1 and sets `errno` to the corresponding value;

- `[EBADF]`= The fd argument is not a valid file descriptor.
- `[ENOTTY]`= The file associated with fd is not a terminal.

If any of the following conditions occur, the `tcdrain()` function returns -1 and sets `errno` to the corresponding value:

- `[EBADF]`= The fd argument is not a valid file descriptor.
- `[EINVAL]`= The queue_selector argument is not a proper value.
- `[ENOTTY]`= The file associated with fd is not a terminal.

If any of the following conditions occur, the `tcflow()` function returns -1 and sets `errno` to the corresponding value:

- `[EBADF]`= The fd argument is not a valid file descriptor.
- `[EINVAL]`= The action argument is not a proper value.
- `[ENOTTY]`= The file associated with fd is not a terminal.

8. Language-Specific Services for the C Programming Language

8.1 Referenced C Language Routines

8.1.2 Extensions to setlocale() Function

8.1.2.2 Description

Locale values recognized by **`setlocale()`** are "`C`" and "`POSIX`.1"

For `setlocale()`, the default value is "`C`" for the following required categories:

- `LC_CTYPE`
- `LC_COLLATE`
- `LC_TIME`
- `LC_NUMERIC`
- `LC_MONETARY`

If no non-null environment variable (`LC_ALL`, `LANG`, or the environment variable corresponding to the category being set) is present to supply a value for locale, `setlocale(category, "")` sets the specified locale category to "`C`."

When the first character in the **`TZ`** environment variable is a colon, the characters that follow the colon are interpreted as follows:

Characters beyond the colon are ignored.

A `TZ` environment variable that begins with a colon is treated the same as a null `TZ` environment variable.

8.2 C Language Input/Output Functions

8.2.1 Map a Stream Pointer to a File Descriptor

8.2.1.4 Errors

No errors are detected for **`fileno()`**.

8.2.2 Open a Stream on a File Descriptor

8.2.2.4 Errors

If unsuccessful, **`fdopen()`** returns a NULL pointer.

8.2.3 Interactions of Other FILE-Type C Functions

For function calls involving two or more file handles, when the actions are coordinated as described in section 8.2.3 of POSIX.1 the output is seen exactly once under all conditions.

8.3 Other C Language Functions

8.3.2 Set Time Zone

8.3.2.2 Description

When the environment variable `TZ` is not set, the LynxOS kernel is queried (using the **`gettimeofday()`** system call) to provide any time zone information it may have.

9. System Databases

9.1 System Databases

If the initial working directory field is null, that field is interpreted as follows:

The root directory (`/`) is used as a default.

If the user program field is null, that field is interpreted as follows:

The shell program `/bin/sh` is used as a default.

10. Data Interchange Format

10.1 Archive/Interchange File Format

The format-reading and format-creating utilities are named **`tar`** and **`cpio`**. See the commands manual for a description of these utilities and the interfaces to them.

10.1.1 Extended tar Format

LynxOS supports the use of characters outside the portable filename character set in names for files, users, and/or groups. The following encoding is provided for interchange purposes:

The `tar` utility does not perform any encoding on the characters used for file names, user names, and group names.

If a file name is found on the medium that would create an invalid file name on the system, the data from the file is not stored on the file hierarchy and a warning is issued.

10.1.2 Extended cpio Format

10.1.2.1 cpio Header

For character or block special files, `c_rdev` contains the device's major and minor numbers.

10.1.2.2 cpio File Name

If a file name is found on the medium that would create an invalid file name, the data from the file is not stored on the file hierarchy and a warning is issued.

10.1.3 Multiple Volumes

The format-creating utilities for the `ustar` and `cpio` formats determine what file to read or write for the next volume of a multivolume archive as follows:

ustar Format

1. When the `tar` utility detects end-of-media, it closes the device and prompts the user to change media and press the **Enter** key to continue.
2. After the user presses **Enter**, the `tar` utility opens the same device and continues to read/write the archive where it left off.

cpio Format

1. When the `cpio` utility detects end-of-media, it closes the device and prompts the user with one of two possible messages, depending on how the `cpio` utility is accessing the archive.
2. If standard input (for extracting) or standard output (for creating) is being used, the `cpio` utility asks the user to enter the name of a device on which to continue the archive extraction/creation. Otherwise, if neither standard input nor standard output is being used, the `cpio` utility simply prompts the user to change media and press **Return**. The same device is used to continue the archive.

11. Synchronization

LynxOS does not require the name of a semaphore to begin with a `/` character. No special interpretation of `/` characters within the name is done.

`sem_open()` returns failure with an error code of `EINVAL` if `O_CREAT` was specified in the flags argument and the initial value of the semaphore is greater than `SEM_VALUE_MAX`.

11.1 Semaphore Characteristics

LynxOS supports the POSIX semaphores option.

11.2.3 Initialize/Open a Named Semaphore

11.2.3.2. Description

When a named semaphore is created, the group ID of the semaphore is set to the effective group ID of the process.

LynxOS does not place the POSIX semaphores in the file system, so all slashes are not relevant to locating the named semaphore. It is a purely string-compare operation.

11.3.1 Mutex Initialization Attributes

`pthread_mutexattr_t`

The default attributes for a mutex when it is statically declared or when it is initialized via **`pthread_mutex_init()`** called with a `NULL pthread_mutexattr_t *`, or with a pointer to a `pthread_mutexattr_t` that has been initialized via `pthread_mutexattr_init()` is as follows:

- The protocol attribute is `PTHREAD_PRIO_INHHERIT`.
- The prioceiling attribute is irrelevant, since the protocol is not `PTHREAD_PRIO_PROTECT`.
- The process-shared attribute is `PTHREAD_PROCESS_PRIVATE`. However, this attribute is not considered when creating a mutex, since all mutexes can be shared between processes under LynxOS.

Attempts to operate on a `pthread_mutexattr_t` that has either not been initialized (statically or via `pthread_mutexattr_init()`) or uninitialized via `pthread_mutexattr_destroy()` results in an error.

11.4.1 Condition Variable Initialization Attributes

`pthread_condattr_t`

The default attribute for a condition variable, whether it is statically initialized, created via **`pthread_cond_init()`** with a `NULL pthread_condattr_t *`, or with a pointer to a `pthread_condattr_t` that has been initialized via `pthread_condattr_init()` is as follows:

The *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, however, this attribute is not considered when creating a condition variable, since all condition variables can be shared between processes under LynxOS.

Attempts to operate on a `pthread_condattr_t` that has either not been initialized (statically or via `pthread_condattr_init()`) or uninitialized via `pthread_condattr_destroy()` results in an error.

12. Memory Management

When a region of memory is locked, its virtual-to-physical mapping is fixed.

12.1.1 Lock/Unlock the Address Space of a Process

When a process calls **`mlockall`** with an argument of **`MCL_FUTURE`**, and the amount of memory being attempted to lock exceeds the amount of physical memory, the allocation request fails.

12.1.2 Lock/Unlock a Range of Process Address Space

There are no alignment restrictions on the address passed to **`mlock()`**.

12.2.4 Memory Object Synchronization

There are no alignment restrictions on addresses passed to **`msync()`**, but the current implementation does not support unmapped files.

12.3.1 Memory Mapped File Restrictions

The `mmap` interface does not support NFS mounted files.

13. Execution Scheduling

13.2.3 SCHED_OTHER

The **`SCHED_OTHER`** execution scheduling policy is a priority quantum scheduler.

13.3.1 Set Scheduling Parameters

13.3.1.2 Description

The conditions under which a process may change another process's scheduling parameters is if the calling process has appropriate privileges, or the effective UID of the target is equal to that of the requesting process.

There are no restrictions on a thread setting its own scheduling policy.

13.4.2 Scheduling Contention Scope

Any scheduling contention scope value is treated as scheduling contention scope `PTHREAD_SCOPE_SYSTEM`. That is, all threads compete for processors with all other threads in the system.

13.4.3 Scheduling Allocation Domain

There is no real notion of a scheduling allocation domain, since all supported platforms under LynxOS are uniprocessor systems.

14. Clocks and Timers

14.1.4 Manifest Constants

The resolution of **`nanosleep()`** is the same as that of the **`clock_gettime()`** with a clock ID of `CLOCK_REALTIME`.

14.2.1 Clocks

14.2.1.2 Description

Supported clock IDs other than `CLOCK_REALTIME` is a configuration option.

There is no effect on armed per-process timers if a call to **`clock_settime()`** is done.

The level of privilege to set a clock via `clock_settime()` is that of appropriate privilege.

14.2.4 Per-Process Timers

14.2.4.2 Description

The value of **DELAYTIMER_MAX** is INT_MAX.

15. Message Passing

15.2.1 Open a Message Queue

15.2.1.2 Description

There is no requirement for names passed to **mq_open()** to start with a /. Furthermore, the conditions that mq_open() returns failure with an error code of EINVAL is of the **attr** argument is invalid.

15.2.7 Set Message Queue Attributes

15.2.7.2 Description

When **_POSIX_SOURCE** is defined, the only flag that mq_setattr() understands is O_NONBLOCK.

16. Thread Management

16.2.1 Thread Creation Attributes

pthread_attr_t

The default attributes for a created thread, when **pthread_create()** is called with a pointer to an initialized pthread_attr_t (via pthread_attr_init()) is as follows:

- The stacksize attribute is 4 Mb.
- The stackaddr attribute is not supported since {POSIX_THREAD_ATTR_STACKADDR} is not defined.

- The inheritsched attribute is `PTHREAD_INHERIT_SCHED`.
- The `priority` attribute is irrelevant, since the priority is inherited from the caller of `pthread_create ()`.
- The `schedpolicy` attribute is `SCHED_FIFO`.
- The `detachstate` attribute is `PTHREAD_CREATE_JOINABLE`.
- The `contentionscope` attribute is `PTHREAD_SCOPE_SYSTEM`. However, this attribute is not considered when creating a thread under LynxOS, because all threads compete for resources on a system-wide basis.
- The `guardsize` attribute (from P1003.1j Draft 7) is `PAGESIZE`.

Attempts to operate on a `pthread_attr_t` that has either not been initialized (statically or via `pthread_attr_init()`) or uninitialized via `pthread_attr_destroy()` results in an error.

18. Thread Cancellation

Previous releases of LynxOS used a special signal to implement thread cancellation. This can lead to problems if an application blocks or ignores the special signal. LynxOS 4.0 implements cancellation as a mechanism similar to, but distinct from a special signal to avoid these potential problems.

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W