RTEMS POSIX API User’s Guide

Edition 4.10.99.0, for RTEMS 4.10.99.0

24 February 2013

RTEMS POSIX API 用户指南

版本：4.10.99.0，适用于 RTEMS 4.10.99.0

2013 年 2 月 24 日

（译注：本文档的英文原版经常更新，但本页上所注的日期并不更新，本次翻译所依据的英文原版是 20130610 版。）

翻译：Tony Zhu　　bizasia@126.com

On-Line Applications Research Corporation

在线应用研究公司

On-Line Applications Research Corporation
TEXinfo 2012-11-08.11
在线应用研究公司
TEXinfo 2012-11-08.11

The RTEMS Project is hosted at http: / /www .rtems .com. Any inquiries concerning RTEMS, its related support components, its documentation, or any custom services for RTEMS should be directed to the contacts listed on that site. A current list of RTEMS Support Providers is at http://www.rtems.com/oarsupport.
RTEMS 项目托管在 http://www .rtems .com。任何涉及 RTEMS 的查询、相关的支持组件、文档或任何定制服务，可以通过在该网站上的通讯录联系。一个当前的 RTEMS 支持提供者列表在 http://www.rtems.com/oarsupport。

Table of Contents

目录

Preface

前言

This is the User’s Guide for the POSIX API support provided in RTEMS.
这是在 RTEMS 中提供的 POSIX API 支持的用户指南。

The functionality described in this document is based on the following standards:
本文档中描述的功能基于以下标准：

- POSIX 1003.1b-1993.
- POSIX 1003.1h/D3.
- Open Group Single UNIX Specification.

Much of the POSIX API standard is actually implemented in the Cygnus Newlib ANSI C Library. Please refer to documentation on Newlib for more information on the functionality it supplies.
许多 POSIX API 标准实际上是在 Cygnus Newlib ANSI C 库中实现的。有关它所提供的功能的更多信息，请参阅 Newlib 上的文档。

This manual is still under construction and improvements are welcomed from users.
本手册仍然欢迎来自用户的建设和改善。

Acknowledgements

致谢

The Institute of Electrical and Electronics Engineers, Inc and The Open Group, have given us permission to reprint portions of their documentation.
电气和电子工程师协会、公司和开放组给予我们许可重印他们文档的某些部分。

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2004 Edition, Standard for Information Technology Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright 2001-2004 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html.

1 Process Creation and Execution Manager

1 进程创建和执行管理器

1.1 Introduction

1.1 简介

The process creation and execution manager provides the functionality associated with the creation and termination of processes.
进程创建和执行管理器提供与进程的创建和终止相关联的功能。

The directives provided by the process creation and execution manager are:
由进程创建和执行管理器提供的指令有：

- fork - Create a Process
- execl - Execute a File
- execv - Execute a File
- execle - Execute a File
- execve - Execute a File
- execlp - Execute a File
- execvp - Execute a File
- pthread_atfork - Register Fork Handlers
- wait - Wait for Process Termination
- waitpid - Wait for Process Termination
- _exit - Terminate a Process

- fork - 创建进程
- execl - 执行一个文件
- execv - 执行一个文件
- execle - 执行一个文件
- execve - 执行一个文件
- execlp - 执行一个文件
- execvp - 执行一个文件
- pthread_atfork - 寄存器分叉处理程序
- wait - 等待进程终止
- waitpid - 等待进程终止
- _exit - 终止进程

1.2 Background

1.2 背景知识

POSIX process functionality can not be completely supported by RTEMS. This is because RTEMS provides no memory protection and implements a single process, multi-threaded execution model. In this light, RTEMS provides none of the routines that are associated with the creation of new processes. However, since the entire RTEMS application (e.g. executable) is logically a single POSIX process, RTEMS is able to provide implementations of many operations on processes. The rule of thumb is that those routines provide a meaningful result. For example, getpid() returns the node number.
POSIX 进程功能不能被 RTEMS 完全支持。这是因为 RTEMS 没有提供内存保护和实现单进程、多线程的执行模式。有鉴于此，RTEMS 没有提供一个与创建新进程相关联的例程。然而，由于整个 RTEMS 应用程序（如可执行文件）逻辑上是单个 POSIX 进程，RTEMS 能够提供在进程上的许多操作的实现。经验法则是那些例程提供有意义的结果。例如，getpid()返回节点号。

1.3 Operations

1.3 操作

The only functionality method defined by this manager which is supported by RTEMS is the _exit service. The implementation of _exit shuts the application down and is equivalent to invoking either exit or rtems_shutdown_executive.
由 RTEMS 支持的此管理器所定义的唯一功能性方法是_exit 服务。_exit 的实现关闭应用程序，相当于调用 exit 或 rtems_shutdown_executive。

1.4 Directives

1.4 指令

This section details the process creation and execution manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了进程创建和执行管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

1.4.1 fork - Create a Process

1.4.1 fork - 创建进程

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>

int fork( void );
```

STATUS CODES: 状态码：

ENOSYS	This routine is not supported by RTEMS.
ENOSYS	RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.2 execl - Execute a File

1.4.2 execl - 执行一个文件

CALLING SEQUENCE: 调用序列：

```
int execl(
    const char *path,
    const char *arg,
    ...
);
```

STATUS CODES: 状态码：

ENOSYS This routine is not supported by RTEMS.
ENOSYS RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.3 execv - Execute a File

1.4.3 execv - 执行一个文件

CALLING SEQUENCE: 调用序列：

```
int execv(
   const char *path,
   char const *argv[],
   ...
);
```

STATUS CODES: 状态码：

ENOSYS	This routine is not supported by RTEMS.
ENOSYS	RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.4 execle - Execute a File

1.4.4 execle - 执行一个文件

CALLING SEQUENCE: 调用序列：

```
int execle(
   const char *path,
   const char *arg,
   ...
);
```

STATUS CODES: 状态码：

ENOSYS This routine is not supported by RTEMS.
ENOSYS RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.5 execve - Execute a File

1.4.5 execve - 执行一个文件

CALLING SEQUENCE: 调用序列：

```
int execve(
   const char *path,
   char *const argv[],
   char *const envp[]
);
```

STATUS CODES: 状态码：

ENOSYS This routine is not supported by RTEMS.
ENOSYS RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.6 execlp - Execute a File

1.4.6 execlp - 执行一个文件

CALLING SEQUENCE: 调用序列：

```
int execlp(
    const char *file,
    const char *arg,
    ...
);
```

STATUS CODES: 状态码：

ENOSYS This routine is not supported by RTEMS.
ENOSYS RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.7 execvp - Execute a File

1.4.7 execvp - 执行一个文件

CALLING SEQUENCE: 调用序列：

```
int execvp(
   const char *file,
   char *const argv[]
   ...
);
```

STATUS CODES: 状态码：

ENOSYS This routine is not supported by RTEMS.
ENOSYS RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.8 pthread_atfork - Register Fork Handlers

1.4.8 pthread_atfork - 寄存器分叉处理程序

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>

int pthread_atfork(
   void (*prepare)(void),
   void (*parent)(void),
   void   (*child)(void)
);
```

STATUS CODES: 状态码：

ENOSYS This routine is not supported by RTEMS.
ENOSYS RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.9 wait - Wait for Process Termination

1.4.9 wait - 等待进程终止

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include   <sys/wait.h>

int wait(
   int *stat_loc
);
```

STATUS CODES: 状态码：

ENOSYS This routine is not supported by RTEMS.
ENOSYS RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.10 waitpid - Wait for Process Termination

1.4.10 waitpid - 等待进程终止

CALLING SEQUENCE: 调用序列：

```
int wait(
   pid_t   pid,
   int    *stat_loc,
   int    options
);
```

STATUS CODES: 状态码：

ENOSYS	This routine is not supported by RTEMS.
ENOSYS	RTEMS 不支持此例程。

DESCRIPTION: 描述：

This routine is not supported by RTEMS.
RTEMS 不支持此例程。

NOTES: 注意事项：

NONE
无

1.4.11 _exit - Terminate a Process

1.4.11 _exit - 终止进程

CALLING SEQUENCE: 调用序列：

```
void _exit(
   int status
);
```

STATUS CODES: 状态码：

NONE
无

DESCRIPTION: 描述：

The _exit() function terminates the calling process.
_exit()函数终止正在调用的进程。

NOTES: 注意事项：

In RTEMS, a process is equivalent to the entire application on a single processor. Invoking this service terminates the application.
在 RTEMS 中，进程相当于在单个处理器上的整个应用程序。调用此服务终止应用程序。

2 Signal Manager

2 信号管理器

2.1 Introduction

2.1 简介

The signal manager provides the functionality associated with the generation, delivery, and management of process-oriented signals.
信号管理器提供与产生、传送和面向过程的信号管理相关联的功能。

The directives provided by the signal manager are:
由信号管理器提供的指令有：

- sigaddset - Add a Signal to a Signal Set
- sigdelset - Delete a Signal from a Signal Set
- sigfillset - Fill a Signal Set
- sigismember - Is Signal a Member of a Signal Set
- sigemptyset - Empty a Signal Set
- sigaction - Examine and Change Signal Action
- pthread_kill - Send a Signal to a Thread
- sigprocmask - Examine and Change Process Blocked Signals
- pthread_sigmask - Examine and Change Thread Blocked Signals
- kill - Send a Signal to a Process
- sigpending - Examine Pending Signals
- sigsuspend - Wait for a Signal
- pause - Suspend Process Execution
- sigwait - Synchronously Accept a Signal
- sigwaitinfo - Synchronously Accept a Signal
- sigtimedwait - Synchronously Accept a Signal with Timeout
- sigqueue - Queue a Signal to a Process
- alarm - Schedule Alarm
- ualarm - Schedule Alarm in Microseconds

- sigaddset - 添加信号到信号集
- sigdelset - 从信号集删除信号
- sigfillset - 填充信号集
- sigismember - 信号是信号集的成员吗？

- sigemptyset - 清空信号集
- sigaction - 检查和更改信号的操作
- pthread_kill - 发送信号到线程
- sigprocmask - 检查和更改进程阻塞的信号
- pthread_sigmask - 检查和更改线程阻塞的信号
- kill - 发送信号到进程
- sigpending - 检查挂起的信号
- sigsuspend - 等待一个信号
- pause - 推迟进程执行
- sigwait - 同步接受信号
- sigwaitinfo - 同步接受信号
- sigtimedwait - 同步接受具有超时的信号
- sigqueue - 排队信号到进程
- alarm - 调度警报
- ualarm - 以微秒为单位的警报

2.2 Background

2.2 背景知识

2.2.1 Signals

2.2.1 信号

POSIX signals are an asynchronous event mechanism. Each process and thread has a set of signals associated with it. Individual signals may be enabled (e.g. unmasked) or blocked (e.g. ignored) on both a per-thread and process level. Signals which are enabled have a signal handler associated with them. When the signal is generated and conditions are met, then the signal handler is invoked in the proper process or thread context asynchronous relative to the logical thread of execution.
POSIX 信号是异步事件机制。每个进程和线程有一组与它相关联的信号。可以在每线程和进程级别上启用（如非屏蔽的）或阻塞（如忽略的）个别信号。已启用的信号有与它们相关联的信号处理程序。当产生信号并且符合条件时，在相对于逻辑执行线程异步的适当的进程或线程上下文中调用信号处理程序。

If a signal has been blocked when it is generated, then it is queued and kept pending until the thread or process unblocks the signal or explicitly checks for it. Traditional, non-real-time POSIX signals do not queue. Thus if a process or thread has blocked a particular signal, then multiple occurrences of that signal are recorded as a single

occurrence of that signal.
如果信号在它产生时已被阻塞，则它被排队并保持挂起，直到线程或进程解锁该信号或显式地检查它。传统的、非实时的 POSIX 信号不排队。因此如果进程或线程已阻塞特定的信号，则该信号的多次出现被记录为该信号的一次出现。

One can check for the set of outstanding signals that have been blocked. Services are provided to check for outstanding process or thread directed signals.
你可以检查已阻塞的未解决信号集。提供服务以检查未解决进程或线程的指向信号。

2.2.2 Signal Delivery

2.2.2 信号传送

Signals which are directed at a thread are delivered to the specified thread.
指向线程的信号被传送到指定的线程。

Signals which are directed at a process are delivered to a thread which is selected based on the following algorithm:
指向进程的信号被传送到根据以下算法选择的线程。

1. If the action for this signal is currently SIG_IGN, then the signal is simply ignored.
1. 如果对此信号的操作是当前的 SIG_IGN，则简单地忽略该信号。

2. If the currently executing thread has the signal unblocked, then the signal is delivered to it.
2. 如果当前执行的线程有信号未阻塞，则传送该信号给它。

3. If any threads are currently blocked waiting for this signal (sigwait()), then the signal is delivered to the highest priority thread waiting for this signal.
3. 如果有任何线程当前被阻塞以等待此信号（sigwait()），则该信号被传送给等待此信号的最高优先级的线程。

4. If any other threads are willing to accept delivery of the signal, then the signal is delivered to the highest priority thread of this set. In the event, multiple threads of the same priority are willing to accept this signal, then priority is given first to ready threads, then to threads blocked on calls which may be interrupted, and finally to threads blocked on non-interruptible calls.
4. 如果有任何其他线程乐意接受该信号的传送，则该信号被传送到此集合的最

高优先级线程。如果，相同优先级的多个线程乐意接受此信号，则优先权首先被给予就绪的线程，然后线程阻塞在可以被中断的调用上，最后线程阻塞在不可中断的调用上。

5. In the event the signal still can not be delivered, then it is left pending. The first thread to unblock the signal (sigprocmask() or pthread_sigprocmask()) or to wait for this signal (sigwait()) will be the recipient of the signal.
5. 如果该信号还不能被传送，则它保持挂起。要解锁该信号（sigprocmask()或pthread_sigprocmask()）或等待此信号（sigwait()）的第一个线程将是该信号的接受者。

2.3 Operations

2.3 操作

2.3.1 Signal Set Management

2.3.1 信号集管理

Each process and each thread within that process has a set of individual signals and handlers associated with it. Services are provided to construct signal sets for the purposes of building signal sets – type sigset_t – that are used to provide arguments to the services that mask, unmask, and check on pending signals.
每个进程和该进程内的每个线程有与它相关联的一组个别信号和处理程序。为了建立信号集的目的，提供建立信号集的服务，类型 sigset_t 用于提供参数给屏蔽、解除屏蔽和检查挂起信号的服务。

2.3.2 Blocking Until Signal Generation

2.3.2 阻塞直到信号产生

A thread may block until receipt of a signal. The "sigwait" and "pause" families of functions block until the requested signal is received or if using sigtimedwait() until the specified timeout period has elapsed.
线程可以阻塞直到收到信号。“sigwait”和“sigwait”的函数家族阻塞，直到收到所请求的信号，或如果使用 sigtimedwait()，直到指定的超时周期已流逝。

2.3.3 Sending a Signal

2.3.3 发送信号

This is accomplished via one of a number of services that sends a signal to either a process or thread. Signals may be directed at a process by the service kill() or at a thread by the service pthread_kill()
这是通过许多服务中的一个发送信号到进程或线程来完成的。信号可以由服务 kill()指向一个进程或由服务 pthread_kill()指向一个线程。

2.4 Directives

2.4 指令

This section details the signal manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了信号管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

2.4.1 sigaddset - Add a Signal to a Signal Set

2.4.1 sigaddset - 添加信号到信号集

CALLING SEQUENCE: 调用序列：

```
#include    <signal.h>

int sigaddset(
   sigset_t *set,
   int    signo
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回0，否则返回-1并设置errno以表示错误。errno可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

This function adds the signal signo to the specified signal set.
这个函数添加信号 signo 到指定的信号集。

NOTES: 注意事项：

The set must be initialized using either sigemptyset or sigfillset before using this function.
在使用此函数之前必须使用 sigemptyset 或 sigfillset 初始化该集合。

2.4.2 sigdelset - Delete a Signal from a Signal Set

2.4.2 sigdelset - 从信号集删除信号

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigdelset(
   sigset_t *set,
   int   signo
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

This function deletes the signal specified by signo from the specified signal set.
该函数从指定的信号集删除由 signo 指定的信号集。

NOTES: 注意事项：

The set must be initialized using either sigemptyset or sigfillset before using this function.
在使用此函数之前必须使用 sigemptyset 或 sigfillset 初始化该集合。

2.4.3 sigfillset - Fill a Signal Set

2.4.3 sigfillset - 填充信号集

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigfillset(
    sigset_t *set
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

This function fills the specified signal set such that all signals are set.
这个函数填充指定的信号集，这样所有信号被设置。

2.4.4 sigismember - Is Signal a Member of a Signal Set

2.4.4 sigismember - 信号是信号集的成员吗？

CALLING SEQUENCE: 调用序列：

```
#include    <signal.h>

int sigismember(
  const sigset_t *set,
    int    signo
);
```

STATUS CODES: 状态码：

The function returns either 1 or 0 if completed successfully, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
如果成功完成，该函数返回 1 或 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL	Invalid argument passed.
EINVAL	无效的参数传递。

DESCRIPTION: 描述：

This function returns returns 1 if signo is a member of set and 0 otherwise.
如果 signo 是 set 的成员，此函数返回 1，否则返回 0。

NOTES: 注意事项：

The set must be initialized using either sigemptyset or sigfillset before using this function.
在使用此函数之前必须使用 sigemptyset 或 sigfillset 初始化该集合。

2.4.5 sigemptyset - Empty a Signal Set

2.4.5 sigemptyset - 清空信号集

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigemptyset(
   sigset_t *set
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

This function initializes an empty signal set pointed to by set.
这个函数初始化由 set 指向的空信号集。

2.4.6 sigaction - Examine and Change Signal Action

2.4.6 sigaction - 检查和更改信号的操作

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigaction(
  int  sig,
  const struct sigaction *act,
    struct sigaction  *oact
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

ENOTSUP Realtime Signals Extension option not supported.
ENOTSUP 不支持实时信号扩展选项。

DESCRIPTION: 描述：

If the argument act is not a null pointer, it points to a structure specifying the action to be associated with the specified signal. If the argument oact is not a null pointer, the action previously associated with the signal is stored in the location pointed to by the argument oact. If the argument act is a null pointer, signal handling is unchanged; thus, the call can be used to enquire about the current handling of a given signal.
如果参数 act 不是 null 指针，它指向与指定信号相关联的指定操作的结构。如果参数 oact 不是 null 指针，先前与该信号相关联的操作被存储在参数 oact 所指向的位置。如果参数 act 是 null 指针，信号处理保持不变；因此，调用可以用于询问给定信号的当前处理。

The structure sigaction has the following members:
结构 sigaction 有以下成员：

void(*)(int) sa_handler
Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
指向信号捕捉函数，或宏 SIG_IGN 或 SIG_DFL 之一的指针。

sigset_t sa_mask Additional set of signals to be blocked during execution of signal-catching function.
sigset_t sa_mask 在信号捕捉函数的执行期间被阻塞的附加信号集。

int sa_flags Special flags to affect behavior of signal.
int sa_flags 影响信号行为的特殊标志。

void(*)(int, siginfo_t*, void*) sa_sigaction
Alternative pointer to a signal-catching function.
指向信号捕捉函数的替代指针。

sa_handler and sa_sigaction should never be used at the same time as their storage may overlap.
sa_handler 和 sa_sigaction 从不应在同一时间用作它们可能重叠的存储。

If the SA_SIGINFO flag (see below) is set in sa_flags, the sa_sigaction field specifies a signal-catching function, otherwise sa_handler specifies the action to be associated with the signal, which may be a signal-catching function or one of the macros SIG_IGN or SIG_DFN.
如果在 sa_flags 中设置了 SA_SIGINFO 标志（见下文），sa_sigaction 字段指定信号捕捉函数，否则 sa_handler 指定与信号相关联的操作，可能是信号捕捉函数，或宏 SIG_IGN 或 SIG_DFN 之一。

The following flags can be set in the sa_flags field:
可以在 sa_flags 字段中设置以下标志：

SA_SIGINFO If not set, the signal-catching function should be declared as void func(int signo) and the address of the function should be set in sa_handler. If set, the signal-catching function should be declared as void func(int signo, siginfo_t* info, void* context) and the address of the function should be set in sa_sigaction.
SA_SIGINFO 如果不设置，信号捕捉函数应被声明为 void func(int signo)，

并应在 sa_handler 中设置该函数的地址。如果设置了，信号捕捉函数应被声明为 void func(int signo, siginfo_t* info, void* context)，并应在 sa_sigaction 中设置该函数的地址。

The prototype of the siginfo_t structure is the following:
siginfo_t 结构的原型如下：

```
typedef struct
{
   int si_signo; /* Signal number */
   int si_code; /* Cause of the signal */
   pid_t si_pid; /* Sending process ID */
   uid_t si_uid; /* Real user ID of sending process */
   void* si_addr; /* Address of faulting instruction */
   int si_status; /* Exit value or signal */
   union sigval
   {
      int sival_int; /* Integer signal value */
      void* sival_ptr; /* Pointer signal value */
   } si_value; /* Signal value */
}
```

NOTES:　注意事项：

The signal number cannot be SIGKILL.
信号码不能为 SIGKILL。

2.4.7 pthread_kill - Send a Signal to a Thread

2.4.7 pthread_kill - 发送信号到线程

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int pthread_kill(
  pthread_t thread,
  int sig
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

ESRCH The thread indicated by the parameter thread is invalid.
ESRCH 参数 thread 所表示的线程无效。

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

This functions sends the specified signal sig to a thread referenced to by thread.
这个函数发送指定的信号 sig 到 thread 所引用的线程。

If the signal code is 0, arguments are validated and no signal is sent.
如果信号码为 0，参数被验证但没有信号被发送。

2.4.8 sigprocmask - Examine and Change Process Blocked Signals

2.4.8 sigprocmask - 检查和更改进程阻塞的信号

CALLING SEQUENCE: 调用序列：

```
#include    <signal.h>

int sigprocmask(
   int    how,
   const sigset_t *set,
   sigset_t    *oset
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL	Invalid argument passed.
EINVAL	无效的参数传递。

DESCRIPTION: 描述：

This function is used to alter the set of currently blocked signals on a process wide basis. A blocked signal will not be received by the process. The behavior of this function is dependent on the value of how which may be one of the following:
这个函数用于改变当前在进程范围上阻塞的信号集。该进程将不会收到阻塞的信号。此函数的行为取决于可能是下列之一的怎样的值：

SIG_BLOCK	The set of blocked signals is set to the union of set and those signals currently blocked.
SIG_BLOCK	阻塞的信号集被设置为集合和这些当前阻塞的信号的联合体。
SIG_UNBLOCK	The signals specific in set are removed from the currently blocked set.
SIG_UNBLOCK	set 中特定的信号从当前阻塞的集合移除。

SIG_SETMASK	The set of currently blocked signals is set to set.
SIG_SETMASK	当前阻塞的信号集被设置为 set。

If oset is not NULL, then the set of blocked signals prior to this call is returned in oset. If set is **NULL**, no change is done, allowing to examine the set of currently blocked signals.
如果 oset 不是 NULL，则在此调用之前阻塞的信号集被返回到 oset 中。如果 set 为 NULL，不做任何改变，允许检查当前阻塞的信号集。

NOTES: 注意事项：

It is not an error to unblock a signal which is not blocked.
解锁未阻塞的信号不是错误。

In the current implementation of RTEMS POSIX API sigprocmask() is technically mapped to pthread_sigmask().
在 RTEMS POSIX API 的当前实现中，sigprocmask()技术上映射到 pthread_sigmask()。

2.4.9 pthread_sigmask - Examine and Change Thread Blocked Signals

2.4.9 pthread_sigmask - 检查和更改线程阻塞的信号

CALLING SEQUENCE: 调用序列：

```
#include    <signal.h>

int pthread_sigmask(
   int    how,
   const sigset_t *set,
   sigset_t    *oset
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

This function is used to alter the set of currently blocked signals for the calling thread. A blocked signal will not be received by the process. The behavior of this function is dependent on the value of how which may be one of the following:
这个函数用于改变调用线程的当前阻塞的信号集。该进程将不会收到阻塞的信号。此函数的行为取决于可能是下列之一的怎样的值：

SIG_BLOCK	The set of blocked signals is set to the union of set and those signals currently blocked.
SIG_BLOCK	阻塞的信号集被设置为集合和这些当前阻塞的信号的联合体。
SIG_UNBLOCK	The signals specific in set are removed from the currently blocked set.

SIG_UNBLOCK	set 中特定的信号从当前阻塞的集合移除。
SIG_SETMASK	The set of currently blocked signals is set to set.
SIG_SETMASK	当前阻塞的信号集被设置为 set。

If oset is not NULL, then the set of blocked signals prior to this call is returned in oset. If set is **NULL**, no change is done, allowing to examine the set of currently blocked signals.
如果 oset 不是 NULL，则在此调用之前阻塞的信号集被返回到 oset 中。如果 set 为 NULL，不做任何改变，允许检查当前阻塞的信号集。

NOTES: 注意事项：

It is not an error to unblock a signal which is not blocked.
解锁未阻塞的信号不是错误。

2.4.10 kill - Send a Signal to a Process

2.4.10 kill - 发送信号到进程

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include   <signal.h>

int   kill(
   pid_t pid,
   int   sig
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

EPERM Process does not have permission to send the signal to any receiving process.
EPERM 进程没有发送信号到任何接收进程的权限。

ESRCH The process indicated by the parameter pid is invalid.
ESRCH 参数 pid 所表示的进程无效。

DESCRIPTION: 描述：

This function sends the signal sig to the process pid.
这个函数发送信号 sig 到进程 pid。

NOTES: 注意事项：

Since RTEMS is a single-process system, a signal can only be sent to the calling process (i.e. the current node).
因为 RTEMS 是单进程系统，信号只能被发送给调用进程（即当前节点）。

2.4.11 sigpending - Examine Pending Signals

2.4.11 sigpending - 检查挂起的信号

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigpending(
const sigset_t *set
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EFAULT Invalid address for set.
EFAULT 无效的 set 地址。

DESCRIPTION: 描述：

This function allows the caller to examine the set of currently pending signals. A pending signal is one which has been raised but is currently blocked. The set of pending signals is returned in set.
这个函数允许调用者检查当前挂起的信号集。挂起信号是已引发但当前被阻塞的信号。在 set 中返回挂起的信号集。

2.4.12 sigsuspend - Wait for a Signal

2.4.12 sigsuspend - 等待一个信号

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigsuspend(
   const sigset_t *sigmask
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINTR Signal interrupted this function.
EINTR 信号中断了此函数。

DESCRIPTION: 描述：

This function temporarily replaces the signal mask for the process with that specified by sigmask and blocks the calling thread until a signal is raised.
这个函数使用 sigmask 所指定暂时替换进程的信号掩码，并阻塞调用线程，直到引发一个信号。

2.4.13 pause - Suspend Process Execution

2.4.13 pause - 推迟进程执行

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int pause( void );
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINTR Signal interrupted this function.
EINTR 信号中断了此函数。

DESCRIPTION: 描述：

This function causes the calling thread to be blocked until an unblocked signal is received.
这个函数引起调用线程被阻塞，直到收到一个非阻塞的信号。

2.4.14 sigwait - Synchronously Accept a Signal

2.4.14 sigwait - 同步接受信号

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigwait(
   const sigset_t *set,
   int    *sig
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

EINTR Signal interrupted this function.
EINTR 信号中断了此函数。

DESCRIPTION: 描述：

This function selects a pending signal based on the set specified in set, atomically clears it from the set of pending signals, and returns the signal number for that signal in sig.
这个函数根据在 set 中指定的集合选择挂起的信号，从挂起的信号集以原子方式清除它，返回该信号在 sig 中的信号码。

2.4.15 sigwaitinfo - Synchronously Accept a Signal

2.4.15 sigwaitinfo - 同步接受信号

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigwaitinfo(
    const sigset_t *set,
    siginfo_t    *info
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EINTR Signal interrupted this function.
EINTR 信号中断了此函数。

DESCRIPTION: 描述：

This function selects a pending signal based on the set specified in set, atomically clears it from the set of pending signals, and returns information about that signal in info.
这个函数根据在 set 中指定的集合选择挂起的信号，从挂起的信号集以原子方式清除它，返回该信号在 info 中的信息。

The prototype of the siginfo_t structure is the following:
siginfo_t 结构的原型如下：

```
typedef struct
{
    int si_signo; /* Signal number */
    int si_code; /* Cause of the signal */
    pid_t si_pid; /* Sending process ID */
    uid_t si_uid; /* Real user ID of sending process */
    void* si_addr; /* Address of faulting instruction */
    int si_status; /* Exit value or signal */
    union sigval
    {
        int sival_int; /* Integer signal value */
        void* sival_ptr; /* Pointer signal value */
    } si_value; /* Signal value */
}
```

2.4.16 sigtimedwait - Synchronously Accept a Signal with Timeout

2.4.16 sigtimedwait - 同步接受具有超时的信号

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigtimedwait(
    const sigset_t *set,
    siginfo_t *info,
    const struct timespec *timeout
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EAGAIN Timed out while waiting for the specified signal set.
EAGAIN 当等待指定的信号集时超时。

EINVAL Nanoseconds field of the timeout argument is invalid.
EINVAL timeout 参数的纳秒字段无效。

EINTR Signal interrupted this function.
EINTR 信号中断了此函数。

DESCRIPTION: 描述：

This function selects a pending signal based on the set specified in set, atomically clears it from the set of pending signals, and returns information about that signal in info. The calling thread will block up to timeout waiting for the signal to arrive.
这个函数根据在 set 中指定的集合选择挂起的信号，从挂起的信号集以原子方式清除它，返回该信号在 info 中的信息。调用线程将阻塞到 timeout，等待该信号到达。

The timespec structure is defined as follows:
timespec 结构的定义如下：

```
struct timespec
{
    time_t tv_sec; /* Seconds */
    long tv_nsec; /* Nanoseconds */
}
```

NOTES:　注意事项：

If timeout is NULL, then the calling thread will wait forever for the specified signal set.
如果 timeout 为 NULL，则调用线程将永远等待指定的信号集。

2.4.17 sigqueue - Queue a Signal to a Process

2.4.17 sigqueue - 排队信号到进程

CALLING SEQUENCE: 调用序列：

```
#include <signal.h>

int sigqueue(
   pid_t  pid,
   int  signo,
   const union sigval value
);
```

STATUS CODES: 状态码：

The function returns 0 on success, otherwise it returns -1 and sets errno to indicate the error. errno may be set to:
该函数成功时返回 0，否则返回-1 并设置 errno 以表示错误。errno 可以被设置为：

EAGAIN No resources available to queue the signal. The process has already queued SIGQUEUE_MAX signals that are still pending at the receiver or the systemwide resource limit has been exceeded.
EAGAIN 没有资源可用于排队信号。进程已排队仍然在接收器上挂起的 SIGQUEUE_MAX 信号，或已超过了全系统的资源限制。

EINVAL The value of the signo argument is an invalid or unsupported signal number.
EINVAL signo 参数的值是无效的或不支持的信号码。

EPERM The process does not have the appropriate privilege to send the signal to the receiving process.
EPERM 进程没有适当的权限发送信号到接收进程。

ESRCH The process pid does not exist.
ESRCH 进程 pid 不存在。

DESCRIPTION: 描述：

This function sends the signal specified by signo to the process pid

这个函数发送由 signo 指定的信号到进程 pid。

The sigval union is specified as:
sigval 联合体被指定为：

```
union sigval
{
   int sival_int; /* Integer signal value */
   void* sival_ptr; /* Pointer signal value */
}
```

NOTES: 注意事项：

Since RTEMS is a single-process system, a signal can only be sent to the calling process (i.e. the current node).
因为 RTEMS 是单进程系统，信号只能被发送给调用进程（即当前节点）。

2.4.18 alarm - Schedule Alarm

2.4.18 alarm - 调度警报

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

unsigned int alarm(
    unsigned int seconds
);
```

STATUS CODES: 状态码：

This call always succeeds.
这个调用总是成功。

If there was a previous alarm() request with time remaining, then this routine returns the number of seconds until that outstanding alarm would have fired. If no previous alarm() request was outstanding, then zero is returned.
如果有具有剩余时间的先前的 alarm()请求，则此例程返回直到该未解决的警报解除为止的秒数。如果没有先前的 alarm()请求未解决，则返回 0。

DESCRIPTION: 描述：

The alarm() service causes the SIGALRM signal to be generated after the number of seconds specified by seconds has elapsed.
alarm()服务导致在由 seconds 指定的秒数流逝之后产生 SIGALRM 信号。

NOTES: 注意事项：

Alarm requests do not queue. If alarm is called while a previous request is outstanding, the call will result in rescheduling the time at which the SIGALRM signal will be generated.
警报请求不排队。如果当先前的请求未解决时调用警报，该调用将导致重新调度产生 SIGALRM 信号的时间。

If the notification signal, SIGALRM, is not caught or ignored, the calling process is terminated.
如果没有捕捉或忽略通知信号 SIGALRM，则终止调用进程。

2.4.19 ualarm - Schedule Alarm in Microseconds

2.4.19 ualarm - 以微秒为单位的警报

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

useconds_t ualarm(
   useconds_t useconds,
   useconds_t    interval
);
```

STATUS CODES: 状态码：

This call always succeeds.
这个调用总是成功。

If there was a previous ualarm() request with time remaining, then this routine returns the number of seconds until that outstanding alarm would have fired. If no previous alarm() request was outstanding, then zero is returned.
如果有具有剩余时间的先前的 ualarm()请求，则此例程返回直到该未解决的警报解除为止的秒数。如果没有先前的 alarm()请求未解决，则返回 0。

DESCRIPTION: 描述：

The ualarm() service causes the SIGALRM signal to be generated after the number of microseconds specified by useconds has elapsed.
ualarm ()服务导致在由 useconds 指定的微秒数流逝之后产生 SIGALRM 信号。

When interval is non-zero, repeated timeout notification occurs with a period in microseconds specified by interval.
当 interval 不为 0 时，使用由 interval 指定的以微秒为单位的周期重复出现超时通知。

NOTES: 注意事项：

Alarm requests do not queue. If alarm is called while a previous request is outstanding, the call will result in rescheduling the time at which the SIGALRM signal will be generated.

警报请求不排队。如果当先前的请求未解决时调用警报，该调用将导致重新调度产生 SIGALRM 信号的时间。

If the notification signal, SIGALRM, is not caught or ignored, the calling process is terminated.
如果没有捕捉或忽略通知信号 SIGALRM，则终止调用进程。

3 Process Environment Manager

3 进程环境管理器

3.1 Introduction

3.1 简介

The process environment manager is responsible for providing the functions related to user and group Id management.
进程环境管理器负责提供有关用户和组 id 管理的函数。

The directives provided by the process environment manager are:
由进程环境管理器提供的指令有：

- getpid - Get Process ID
- getppid - Get Parent Process ID
- getuid - Get User ID
- geteuid - Get Effective User ID
- getgid - Get Real Group ID
- getegid - Get Effective Group ID
- setuid - Set User ID
- setgid - Set Group ID
- getgroups - Get Supplementary Group IDs
- getlogin - Get User Name
- getlogin_r - Reentrant Get User Name
- getpgrp - Get Process Group ID
- setsid - Create Session and Set Process Group ID
- setpgid - Set Process Group ID for Job Control
- uname - Get System Name
- times - Get Process Times
- getenv - Get Environment Variables
- setenv - Set Environment Variables
- ctermid - Generate Terminal Pathname
- ttyname - Determine Terminal Device Name
- ttyname_r - Reentrant Determine Terminal Device Name
- isatty - Determine if File Descriptor is Terminal
- sysconf - Get Configurable System Variables

- getpid - 获得进程 ID
- getppid - 获得父进程 ID
- getuid - 获得用户 ID
- geteuid - 获得有效用户 ID
- getgid - 获得真实组 ID
- getegid - 获得有效组 ID
- setuid - 设置用户 ID
- setgid - 设置组 ID
- getgroups - 获得补充组 ID
- getlogin - 获得用户名
- getlogin_r - 可重入地获得用户名
- getpgrp - 获得进程组 ID
- setsid - 创建会话和进程组 ID
- setpgid - 设置作业控制的进程组 ID
- uname - 获得系统名称
- times - 获得进程时间
- getenv - 获得环境变量
- setenv - 设置环境变量
- ctermid - 生成终端路径名
- ttyname - 确定终端设备名称
- ttyname_r - 可重入地确定终端设备名称
- isatty - 确定文件描述符是否是终端
- sysconf - 获得可配置的系统变量

3.2 Background

3.2 背景知识

3.2.1 Users and Groups

3.2.1 用户和组

RTEMS provides a single process, multi-threaded execution environment. In this light, the notion of user and group is somewhat without meaning. But RTEMS does provide services to provide a synthetic version of user and group. By default, a single user and group is associated with the application. Thus unless special actions are taken, every thread in the application shares the same user and group Id. The initial rationale for providing user and group Id functionality in RTEMS was for the filesystem infrastructure to implement file permission checks. The effective

user/group Id capability has since been used to implement permissions checking by the ftpd server.
RTEMS 提供单进程、多线程的执行环境，就此而论，用户和组的概念有些没有意义。但 RTEMS 提供服务，以提供综合版本的用户和组。默认情况下，单个用户和组与应用程序相关联。因此除非采取特殊操作，应用程序中的每个线程共享相同的用户和组 id。在 RTEMS 中提供用户和组 id 功能的最初理由是为文件系统基础架构实现文件权限检查。有效的用户/组 id 后来已被用于实现 ftpd 服务器的权限检查。

In addition to the "real" user and group Ids, a process may have an effective user/group Id. This allows a process to function using a more limited permission set for certain operations.
除了“真实的”用户和组 id，进程可能有有效的用户/组 id。这允许使用更有限的权限的函数的进程为某一操作设置。

3.2.2 User and Group Names

3.2.2 用户和组的名称

POSIX considers user and group Ids to be a unique integer that may be associated with a name. This is usually accomplished via a file named /etc/passwd for user Id mapping and /etc/groups for group Id mapping. Again, although RTEMS is effectively a single process and thus single user system, it provides limited support for user and group names. When configured with an appropriate filesystem, RTEMS will access the appropriate files to map user and group Ids to names.
POSIX 认为用户和组 id 是唯一的整数，可以与名称相关联。这通常通过为用户 id 映射的名为/etc/passwd 的和为组 id 映射的名为/etc/groups 的文件完成。此外，虽然 RTEMS 实际上是单进程并因而是单用户系统，它为用户和组名称提供有限的支持。当配置了适当的文件系统时，RTEMS 将访问适当的文件以映射用户和组 id 到名称。

If these files do not exist, then RTEMS will synthesize a minimal version so this family of services return without error. It is important to remember that a design goal of the RTEMS POSIX services is to provide useable and meaningful results even though a full process model is not available.
如果这些文件不存在，RTEMS 将合成一个最小版本，所以此服务家族没有错误返回。重要的是要记住，RTEMS POSIX 服务的设计目标是提供可用的和有意义的结果，即使全进程模型不可用。

3.2.3 Environment Variables

3.2.3 环境变量

POSIX allows for variables in the run-time environment. These are name/value pairs that make be dynamically set and obtained by programs. In a full POSIX environment with command line shell and multiple processes, environment variables may be set in one process – such as the shell – and inherited by child processes. In RTEMS, there is only one process and thus only one set of environment variables across all processes.
POSIX 允许运行时环境中的变量。这些是由程序动态设置和获取的名称/值对。在具有命令行 shell 和多进程的完整 POSIX 环境中，环境变量可以在一个进程如 shell 中设置，并由子进程继承。在 RTEMS 中，只有一个进程，因而只有一组跨所有进程的环境变量。

3.3 Operations

3.3 操作

3.3.1 Accessing User and Group Ids

3.3.1 访问用户和组 id

The user Id associated with the current thread may be obtain using the getuid() service. Similarly, the group Id may be obtained using the getgid() service.
与当前线程相关联的用户 id 可以使用 getuid()服务获取。类似地，组 id 可以使用 getgid()服务获取。

3.3.2 Accessing Environment Variables

3.2.2 访问环境变量

The value associated with an environment variable may be obtained using the getenv() service and set using the putenv() service.
与环境变量相关联的值可以使用 getenv()服务获取，使用 putenv()服务设置。

3.4 Directives

3.4 指令

This section details the process environment manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了进程环境管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

3.4.1 getpid - Get Process ID

3.4.1 getpid - 获得进程 ID

CALLING SEQUENCE: 调用序列：

```
int getpid( void );
```

STATUS CODES: 状态码：

The process Id is returned.
返回进程 id。

DESCRIPTION: 描述：

This service returns the process Id.
这个服务返回进程 id。

NOTES: 注意事项：

NONE
无。

3.4.2 getppid - Get Parent Process ID

3.4.2 getppid - 获得父进程 ID

CALLING SEQUENCE: 调用序列：

```
int getppid( void );
```

STATUS CODES: 状态码：

The parent process Id is returned.
返回父进程 id。

DESCRIPTION: 描述：

This service returns the parent process Id.
这个服务返回父进程 id。

NOTES: 注意事项：

NONE
无。

3.4.3 getuid - Get User ID

3.4.3 getuid - 获得用户 ID

CALLING SEQUENCE: 调用序列：

```
int getuid( void );
```

STATUS CODES: 状态码：

The effective user Id is returned.
返回有效的用户 id。

DESCRIPTION: 描述：

This service returns the effective user Id.
这个服务返回有效的用户 id。

NOTES: 注意事项：

NONE
无

3.4.4 geteuid - Get Effective User ID

3.4.4 geteuid - 获得有效用户 ID

CALLING SEQUENCE: 调用序列：

```
int geteuid( void );
```

STATUS CODES: 状态码：

The effective group Id is returned.
返回有效的组 id。

DESCRIPTION: 描述：

This service returns the effective group Id.
这个服务返回有效的组 id。

NOTES: 注意事项：

NONE
无

3.4.5 getgid - Get Real Group ID

3.4.5 getgid - 获得真实组 ID

CALLING SEQUENCE: 调用序列：

```
int getgid( void );
```

STATUS CODES: 状态码：

The group Id is returned.
返回组 id。

DESCRIPTION: 描述：

This service returns the group Id.
这个服务返回组 id。

NOTES: 注意事项：

NONE
无

3.4.6 getegid - Get Effective Group ID

3.4.6 getegid - 获得有效组 ID

CALLING SEQUENCE: 调用序列：

```
int getegid( void );
```

STATUS CODES: 状态码：

The effective group Id is returned.
返回有效组 id。

DESCRIPTION: 描述：

This service returns the effective group Id.
这个服务返回有效组 id。

NOTES: 注意事项：

NONE
无

3.4.7 setuid - Set User ID

3.4.7 setuid - 设置用户 ID

CALLING SEQUENCE: 调用序列：

```
int setuid(
   uid_t uid
);
```

STATUS CODES: 状态码：

This service returns 0.
这个服务返回 0。

DESCRIPTION: 描述：

This service sets the user Id to uid.
这个服务设置用户 id 为 uid。

NOTES: 注意事项：

NONE
无

3.4.8 setgid - Set Group ID

3.4.8 setgid - 设置组 ID

CALLING SEQUENCE: 调用序列：

```
int setgid(
   gid_t   gid
);
```

STATUS CODES: 状态码：

This service returns 0.
这个服务返回 0。

DESCRIPTION: 描述：

This service sets the group Id to gid.
这个服务设置组 id 为 gid。

NOTES: 注意事项：

NONE
无

3.4.9 getgroups - Get Supplementary Group IDs

3.4.9 getgroups - 获得补充组 ID

CALLING SEQUENCE: 调用序列：

```
int getgroups(
   int    gidsetsize,
   gid_t    grouplist[]
);
```

STATUS CODES: 状态码：

NA

DESCRIPTION: 描述：

This service is not implemented as RTEMS has no notion of supplemental groups.
这个服务没有实现，因为 RTEMS 没有补充组的概念。

NOTES: 注意事项：

If supported, this routine would only be allowed for the super-user.
如果支持，只允许将此例程用于超级用户。

3.4.10 getlogin - Get User Name

3.4.10 getlogin - 获得用户名

CALLING SEQUENCE: 调用序列：

```
char *getlogin( void );
```

STATUS CODES: 状态码：

Returns a pointer to a string containing the name of the current user.
返回指向包含当前用户名的字符串的指针。

DESCRIPTION: 描述：

This routine returns the name of the current user.
这个例程返回当前用户名。

NOTES: 注意事项：

This routine is not reentrant and subsequent calls to getlogin() will overwrite the same buffer.
这个例程不是可重入的，后续的 getlogin()调用将覆盖相同的缓冲区。

3.4.11 getlogin_r - Reentrant Get User Name

3.4.11 getlogin_r - 可重入地获得用户名

CALLING SEQUENCE: 调用序列：

```
int getlogin_r(
   char    *name,
   size_t      namesize
);
```

STATUS CODES: 状态码：

EINVAL The arguments were invalid.
EINVAL 参数无效。

DESCRIPTION: 描述：

This is a reentrant version of the getlogin() service. The caller specified their own buffer, name, as well as the length of this buffer, namesize.
这是可重入版本的 getlogin()服务。调用者指定它们自己的缓冲区 name，以及此缓冲区的长度 namesize。

NOTES: 注意事项：

NONE
无

3.4.12 getpgrp - Get Process Group ID

3.4.12 getpgrp - 获得进程组 ID

CALLING SEQUENCE: 调用序列：

```
pid_t getpgrp( void );
```

STATUS CODES: 状态码：

The procress group Id is returned.
返回进程组 id。

DESCRIPTION: 描述：

This service returns the current progress group Id.
这个服务返回当前进程组 id。

NOTES: 注意事项：

This routine is implemented in a somewhat meaningful way for RTEMS but is truly not functional.
这个例程以对 RTEMS 有点意义的方式实现，但不是真正功能。

3.4.13 setsid - Create Session and Set Process Group ID

3.4.13 setsid - 创建会话和进程组 ID

CALLING SEQUENCE: 调用序列：

```
pid_t setsid( void );
```

STATUS CODES: 状态码：

EPERM	The application does not have permission to create a process group.
EPERM	应用程序没有创建进程组的权限。

DESCRIPTION: 描述：

This routine always returns EPERM as RTEMS has no way to create new processes and thus no way to create a new process group.
这个例程总是返回 EPERM，因为 RTEMS 没有方法创建新进程，因而没有方法创建新进程组。

NOTES: 注意事项：

NONE
无

3.4.14 setpgid - Set Process Group ID for Job Control

3.4.14 setpgid - 设置作业控制的进程组 ID

CALLING SEQUENCE: 调用序列：

```
int setpgid(
   pid_t pid,
   pid_t pgid
);
```

STATUS CODES: 状态码：

ENOSYS	The routine is not implemented.
ENOSYS	该例程没有实现。

DESCRIPTION: 描述：

This service is not implemented for RTEMS as process groups are not supported.
这个服务没有为 RTEMS 实现，因为不支持进程组。

NOTES: 注意事项：

NONE
无

3.4.15 uname - Get System Name

3.4.15 uname - 获得系统名称

CALLING SEQUENCE: 调用序列：

```
int uname(
   struct utsname *name
);
```

STATUS CODES: 状态码：

EPERM The provided structure pointer is invalid.
EPERM 提供的结构指针无效。

DESCRIPTION: 描述：

This service returns system information to the caller. It does this by filling in the struct utsname format structure for the caller.
这个服务返回系统信息到调用者。这是通过为调用者填充 struct utsname 格式的结构进行的。

NOTES: 注意事项：

The information provided includes the operating system (RTEMS in all configurations), the node number, the release as the RTEMS version, and the CPU family and model. The CPU model name will indicate the multilib executive variant being used.
提供的信息包括操作系统（在所有配置中的 RTEMS）、节点号、RTEMS 发行版本和 CPU 家族和型号。CPU 型号名称将表示正在使用的 multilib 执行变体。

3.4.16 times - Get process times

3.4.16 times - 获得进程时间

CALLING SEQUENCE: 调用序列：

```
#include <sys/time.h>

clock_t    times(
     struct tms *ptms
);
```

STATUS CODES: 状态码：

This routine returns the number of clock ticks that have elapsed since the system was initialized (e.g. the application was started).
这个例程返回自系统初始化（如启动应用程序）以来已流逝的时钟节拍数。

DESCRIPTION: 描述：

times stores the current process times in ptms. The format of struct tms is as defined in <sys/times.h>. RTEMS fills in the field tms_utime with the number of ticks that the calling thread has executed and the field tms_stime with the number of clock ticks since system boot (also returned). All other fields in the ptms are left zero.
times 在 ptms 中存储当前进程的时间。struct tms 格式在<sys/times.h>中定义。RTEMS 使用已执行的调用线程的节拍数填充字段 tms_utime，使用自系统启动（也可以是返回）以来的时钟节拍数填充 tms_stime。ptms 中的所有其他字段保留为 0。

NOTES: 注意事项：

RTEMS has no way to distinguish between user and system time so this routine returns the most meaningful information possible.
RTEMS 没有方法区分用户时间和系统时间，所以此例程返回可能最有意义的信息。

3.4.17 getenv - Get Environment Variables

3.4.17 getenv - 获得环境变量

CALLING SEQUENCE: 调用序列：

```
char *getenv(
   const char *name
);
```

STATUS CODES: 状态码：

NULL when no match
NULL 当不匹配时

pointer to value when successful
指向值的指针 当成功时

DESCRIPTION: 描述：

This service searches the set of environment variables for a string that matches the specified name. If found, it returns the associated value.
这个服务搜索匹配 name 指定的字符串的环境变量集。如果找到，它返回关联的值。

NOTES: 注意事项：

The environment list consists of name value pairs that are of the form name = value.
环境列表包含名称值对，形式为名称=值。

3.4.18 setenv - Set Environment Variables

3.4.18 setenv -设置环境变量

CALLING SEQUENCE: 调用序列：

```
int setenv(
   const char *name,
   const char *value,
   int    overwrite
);
```

STATUS CODES: 状态码：

Returns 0 if successful and -1 otherwise.
如果成功，返回 0，否则返回-1。

DESCRIPTION: 描述：

This service adds the variable name to the environment with value. If name is not already exist, then it is created. If name exists and overwrite is zero, then the previous value is not overwritten.
这个服务添加变量 name 到环境 value。如果 name 先前不存在，则创建它。如果 name 存在且 overwrite 为 0，则不覆盖先前的值。

NOTES: 注意事项：

NONE
无

3.4.19 ctermid - Generate Terminal Pathname

3.4.19 ctermid - 生成终端路径名

CALLING SEQUENCE: 调用序列：

```
char *ctermid(
   char *s
);
```

STATUS CODES: 状态码：

Returns a pointer to a string indicating the pathname for the controlling terminal.
返回指向表示控制终端路径名的字符串的指针。

DESCRIPTION: 描述：

This service returns the name of the terminal device associated with this process. If s is NULL, then a pointer to a static buffer is returned. Otherwise, s is assumed to have a buffer of sufficient size to contain the name of the controlling terminal.
这个服务返回与此进程相关联的终端设备的名称。如果 s 为 NULL，则返回指向静态缓冲区的指针。否则，假定 s 有足够大小的缓冲区，以包含控制终端的名称。

NOTES: 注意事项：

By default on RTEMS systems, the controlling terminal is /dev/console. Again this implementation is of limited meaning, but it provides true and useful results which should be sufficient to ease porting applications from a full POSIX implementation to the reduced profile supported by RTEMS.
默认情况下，在 RTEMS 系统上，控制终端是/dev/console。此外此实现是有限的意义，但它提供正确的且有用的结果，足以从完整的 POSIX 实现容易地移植应用程序到由 RTEMS 支持的简化的配置文件。

3.4.20 ttyname - Determine Terminal Device Name

3.4.20 ttyname - 确定终端设备名称

CALLING SEQUENCE: 调用序列：

```
char *ttyname(
   int fd
);
```

STATUS CODES: 状态码：

Pointer to a string containing the terminal device name or NULL is returned on any error.
返回指向包含终端设备名称的字符串的指针或任何错误时返回 NULL。

DESCRIPTION: 描述：

This service returns a pointer to the pathname of the terminal device that is open on the file descriptor fd. If fd is not a valid descriptor for a terminal device, then NULL is returned.
这个服务返回指向在文件描述符 fd 上打开的终端设备的路径名的指针。如果 fd 不是对终端设备有效的描述符，则返回 NULL。

NOTES: 注意事项：

This routine uses a static buffer.
这个例程使用静态缓冲区。

3.4.21 ttyname_r - Reentrant Determine Terminal Device Name

3.4.21 ttyname_r - 可重入地确定终端设备名称

CALLING SEQUENCE: 调用序列：

```
int ttyname_r(
   int   fd,
   char *name,
   int   namesize
);
```

STATUS CODES: 状态码：

This routine returns -1 and sets errno as follows:
这个例程返回-1 并设置 errno 如下：

EBADF If not a valid descriptor for a terminal device.
EBADF 如果不是对终端设备有效的描述符。

EINVAL If name is NULL or namesize are insufficient.
EINVAL 如果 name 为 NULL 或 namesize 不足。

DESCRIPTION: 描述：

This service returns the pathname of the terminal device that is open on the file descriptor fd.
这个服务返回在文件描述符 fd 上打开的终端设备的路径名。

NOTES: 注意事项：

NONE
无

3.4.22 isatty - Determine if File Descriptor is Terminal

3.4.22 isatty - 确定文件描述符是否是终端

CALLING SEQUENCE: 调用序列：

```
int isatty(
   int fd
);
```

STATUS CODES: 状态码：

Returns 1 if fd is a terminal device and 0 otherwise.
如果 fd 是终端设备，返回 1，否则返回 0。

DESCRIPTION: 描述：

This service returns 1 if fd is an open file descriptor connected to a terminal and 0 otherwise.
如果 fd 是连接到终端的打开文件描述符，此服务返回 1，否则返回 0。

NOTES: 注意事项：

3.4.23 sysconf - Get Configurable System Variables

3.4.23 sysconf - 获得可配置的系统变量

CALLING SEQUENCE: 调用序列：

```
long sysconf(
   int name
);
```

STATUS CODES: 状态码：

The value returned is the actual value of the system resource. If the requested configuration name is a feature flag, then 1 is returned if the available and 0 if it is not. On any other error condition, -1 is returned.
返回的值是系统资源的实际值。如果请求的配置名称是特征标志，如果可用，返回 1，如果不可用，返回 0。在任何其他错误的情况下，返回-1。

DESCRIPTION: 描述：

This service is the mechanism by which an application determines values for system limits or options at runtime.
这个服务是应用程序在运行时确定系统限制值或选项值的机制。

NOTES: 注意事项：

Much of the information that may be obtained via sysconf has equivalent macros in <unistd.h>. However, those macros reflect conservative limits which may have been altered by application configuration.
可以通过 sysconf 获取的许多信息相当于<unistd.h>中的宏。然而，这些宏反映了可能已被应用程序配置更改的保守的限制。

4 Files and Directories Manager

4 文件和目录管理器

4.1 Introduction

4.1 简介

The files and directories manager is ...
文件和目录管理器是……

The directives provided by the files and directories manager are:
由文件和目录管理器提供的指令有：

- opendir - Open a Directory
- readdir - Reads a directory
- rewinddir - Resets the readdir() pointer
- scandir - Scan a directory for matching entries
- telldir - Return current location in directory stream
- closedir - Ends directory read operation
- getdents - Get directory entries
- chdir - Changes the current working directory
- fchdir - Changes the current working directory
- getcwd - Gets current working directory
- open - Opens a file
- creat - Create a new file or rewrite an existing one
- umask - Sets a file creation mask
- link - Creates a link to a file
- symlink - Creates a symbolic link to a file
- readlink - Obtain the name of the link destination
- mkdir - Makes a directory
- mkfifo - Makes a FIFO special file
- unlink - Removes a directory entry
- rmdir - Delete a directory
- rename - Renames a file
- stat - Gets information about a file.
- fstat - Gets file status
- lstat - Gets file status
- access - Check permissions for a file.

- chmod - Changes file mode
- fchmod - Changes permissions of a file
- chown - Changes the owner and/ or group of a file
- utime - Change access and/or modification times of an inode
- ftruncate - Truncate a file to a specified length
- truncate - Truncate a file to a specified length
- pathconf - Gets configuration values for files
- fpathconf - Get configuration values for files
- mknod - Create a directory

- opendir - 打开目录
- readdir - 读取目录
- rewinddir - 复位 readdir()指针
- scandir - 扫描匹配项的目录
- telldir - 返回目录流中的当前位置
- closedir - 结束目录读取操作
- getdents - 获得目录项
- chdir - 更改当前工作目录
- fchdir - 更改当前工作目录
- getcwd - 获得当前工作目录
- open - 打开文件
- creat - 创建新文件或重写存在的文件
- umask - 设置文件创建掩码
- link - 创建指向文件的链接
- symlink - 创建指向文件的符号连接
- readlink - 获取链接目标的名称
- mkdir - 创建目录
- mkfifo - 创建 FIFO 特殊文件
- unlink - 移除目录项
- rmdir - 删除目录
- rename - 重命名文件
- stat - 获得关于文件的信息
- fstat - 获得文件状态
- lstat - 获得文件状态
- access - 检查文件权限
- chmod - 更改文件模式
- fchmod - 更改文件权限
- chown - 更改文件所有者和/或组。

- utime - 更改 inode 的访问和/或修改时间
- ftruncate - 截断文件到指定的长度
- truncate - 截断文件到指定的长度
- pathconf - 获得文件的配置值
- fpathconf - 获得文件的配置值
- mknod - 创建目录

4.2 Background

4.2 背景知识

4.2.1 Path Name Evaluation

4.2.1 路径名评估

A pathname is a string that consists of no more than PATH_MAX bytes, including the terminating null character. A pathname has an optional beginning slash, followed by zero or more filenames separated by slashes. If the pathname refers to a directory, it may also have one or more trailing slashes. Multiple successive slahes are considered to be the same as one slash.
路径名是包含不超过 PATH_MAX 字节的字符串，包括结束 null 字符。路径名有可选的开始斜杠，接着是由斜杠分隔的零个或多个文件名。如果路径名引用了目录，它可能还有一个或多个后续斜杠。多个连续的斜杠被认为是与一个斜杠一样的。

POSIX allows a pathname that begins with precisely two successive slashes to be interpreted in an implementation-defined manner. RTEMS does not currently recognize this as a special condition. Any number of successive slashes is treated the same as a single slash. POSIX requires that an implementation treat more than two leading slashes as a single slash.
POSIX 允许路径名严格地开始于两个连续的斜杠，以实现定义的方式被解释。RTEMS 当前还不认可这作为特殊情况。任何数量的连续斜杠被视为和单个斜杠一样。POSIX 需要实现将多于两个的前导斜杠视为单个斜杠。

4.3 Operations

4.3 操作

There is currently no text in this section.
当前在本节中没有文本。

4.4 Directives

4.4 指令

This section details the files and directories manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了文件和目录管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

4.4.1 opendir - Open a Directory

4.4.1 opendir - 打开目录

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <dirent.h>

int opendir(
   const char *dirname
);
```

STATUS CODES: 状态码：

EACCES Search permission was denied on a component of the path prefix of dirname, or read permission is denied
EACCES 在 dirname 的路径前缀的组件上拒绝搜索权限，或拒绝读取权限。

EMFILE Too many file descriptors in use by process
EMFILE 太多文件描述符在由进程使用中。

ENFILE Too many files are currently open in the system.
ENFILE 当前在系统中打开了太多文件。

ENOENT Directory does not exist, or name is an empty string.
ENOENT 目录不存在，或 name 是空字符串。

ENOMEM Insufficient memory to complete the operation.
ENOMEM 内存不足以完成操作。

ENOTDIR name is not a directory.
ENOTDIR name 不是目录。

DESCRIPTION: 描述：

This routine opens a directory stream corresponding to the directory specified by the dirname argument. The directory stream is positioned at the first entry.
这个例程打开对应于 dirname 参数所指定的目录的目录流。目录流被放置在第一项。

NOTES:　注意事项：

The routine is implemented in Cygnus newlib.
该例程是在 Cygnus newlib 中实现的。

4.4.2 readdir - Reads a directory

4.4.2 readdir - 读取目录

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <dirent.h>

int readdir(
   DIR *dirp
);
```

STATUS CODES: 状态码：

EBADF	Invalid file descriptor
EBADF	无效的文件描述符

DESCRIPTION: 描述：

The readdir() function returns a pointer to a structure dirent representing the next directory entry from the directory stream pointed to by dirp. On end-of-file, NULL is returned.
readdir()函数返回指向表示来自 dirp 所指向的目录流的下一个目录项的结构 dirent 的指针。在文件末尾时返回 NULL。

The readdir() function may (or may not) return entries for . or .. Your program should tolerate reading dot and dot-dot but not require them.
readdir()函数可能（或不可能）返回.或..的项，你的程序应容忍读取点和点点，但不需要它们。

The data pointed to be readdir() may be overwritten by another call to readdir() for the same directory stream. It will not be overwritten by a call for another directory.
指向 readdir()的数据可能被对相同目录流的 readdir()的另一个调用覆盖。它不会被另一个目录的调用覆盖。

NOTES: 注意事项：

If ptr is not a pointer returned by malloc(), calloc(), or realloc() or has been deallocated with free() or realloc(), the results are not portable and are probably

disastrous.
如果 ptr 不是由 malloc()、calloc()或 realloc()返回的指针，或已使用 free()或 realloc()释放，结果是不可移植和可能造成灾难。

The routine is implemented in Cygnus newlib.
该例程是在 Cygnus newlib 中实现的。

4.4.3 rewinddir - Resets the readdir() pointer

4.4.3 rewinddir - 复位 readdir()指针

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <dirent.h>

void rewinddir(
   DIR *dirp
);
```

STATUS CODES: 状态码：

No value is returned.
不返回值。

DESCRIPTION: 描述：

The rewinddir() function resets the position associated with the directory stream pointed to by dirp. It also causes the directory stream to refer to the current state of the directory.
rewinddir()函数复位与 dirp 所指向的目录流相关联的位置。它还导致目录流引用该目录的当前状态。

NOTES: 注意事项：

~~NONE~~

If dirp is not a pointer by opendir(), the results are undefined.
如果 dirp 不是由 opendir()指向的指针，结果是未定义的。

The routine is implemented in Cygnus newlib.
该例程是在 Cygnus newlib 中实现的。

4.4.4 scandir - Scan a directory for matching entries

4.4.4 scandir - 扫描匹配项的目录

CALLING SEQUENCE: 调用序列：

```
#include <dirent.h>

int scandir(
   const char    *dir,
   struct dirent ***namelist,
   int    (*select)(const struct dirent *),
   int    (*compar)(const struct dirent **, const struct dirent **)
);
```

STATUS CODES: 状态码：

ENOMEM Insufficient memory to complete the operation.
ENOMEM 内存不足以完成操作。

DESCRIPTION: 描述：

The scandir() function scans the directory dir, calling select() on each directory entry. Entries for which select() returns non-zero are stored in strings allocated via malloc(), sorted using qsort() with the comparison function compar(), and collected in array namelist which is allocated via malloc(). If select is NULL, all entries are selected.
scandir()函数扫描目录 dir，在每个目录项上调用 select()。对 select()返回非零的项被存储在通过malloc()分配的字符串中，使用qsort()和比较函数compar()分类，收集在通过 malloc()分配的数组 namelist 中。如果 select 为 NULL，选择所有项。

NOTES: 注意事项：

The routine is implemented in Cygnus newlib.
该例程是在 Cygnus newlib 中实现的。

4.4.5 telldir - Return current location in directory stream

4.4.5 telldir - 返回目录流中的当前位置

CALLING SEQUENCE: 调用序列：

```
#include <dirent.h>

off_t telldir(
   DIR *dir
);
```

STATUS CODES: 状态码：

EBADF Invalid directory stream descriptor dir.
EBADF 无效的目录流描述符 dir。

DESCRIPTION: 描述：

The telldir() function returns the current location associated with the directory stream dir.
telldir()函数返回与目录流 dir 相关联的当前位置。

NOTES: 注意事项：

The routine is implemented in Cygnus newlib.
该例程是在 Cygnus newlib 中实现的。

4.4.6 closedir - Ends directory read operation

4.4.6 closedir - 结束目录读取操作

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <dirent.h>

int   closedir(
   DIR *dirp
);
```

STATUS CODES: 状态码：

EBADF	Invalid file descriptor
EBADF	无效的文件描述符

DESCRIPTION: 描述：

The directory stream associated with dirp is closed. The value in dirp may not be usable after a call to closedir().
与 dirp 相关联的目录流被关闭。在调用 closedir()之后，dirp 中的值可能不可用。

NOTES: 注意事项：

~~NONE~~

The argument to closedir() must be a pointer returned by opendir(). If it is not, the results are not portable and most likely unpleasant.
closedir()的参数必须是由 opendir()返回的指针。如果不是，结果是不可移植和最有可能令人不快。

The routine is implemented in Cygnus newlib.
该例程是在 Cygnus newlib 中实现的。

4.4.7 chdir - Changes the current working directory

4.4.7 chdir - 更改当前工作目录

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int chdir(
   const char *path
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为以下值之一：

EACCES Search permission is denied for a directory in a file’s path prefix.
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified pathname was not a directory when directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

DESCRIPTION: 描述：

The chdir() function causes the directory named by path to become the current working directory; that is, the starting point for searches of pathnames not beginning with a slash.
chdir()函数导致由 path 指定的目录成为当前工作目录；就是说，搜索路径名的起始点不以斜杠开始。

If chdir() detects an error, the current working directory is not changed.
如果 chdir()检测到错误，则不会更改当前工作目录。

NOTES: 注意事项：

NONE
无

4.4.8 fchdir - Changes the current working directory

4.4.8 fchdir - 更改当前工作目录

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int fchdir(
   int fd
);
```

STATUS CODES: 状态码：

出现错误时，此例程返回-1 并设置 errno 为以下值之一：

EACCES Search permission is denied for a directory in a file’s path prefix.
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified pathname was not a directory when directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

DESCRIPTION: 描述：

The fchdir() function causes the directory named by fd to become the current working directory; that is, the starting point for searches of pathnames not beginning with a slash.
fchdir ()函数导致由 fd 指定的目录成为当前工作目录；就是说，搜索路径名的起始点不以斜杠开始。

If fchdir() detects an error, the current working directory is not changed.

如果 fchdir ()检测到错误，则不会更改当前工作目录。

NOTES: 注意事项：

NONE
无

4.4.9 getcwd - Gets current working directory

4.4.9 getcwd - 获得当前工作目录

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int getcwd( void );
```

STATUS CODES: 状态码：

EINVAL Invalid argument
EINVAL 无效的参数

ERANGE Result is too large
ERANGE 结果太大

EACCES Search permission is denied for a directory in a file’s path prefix.
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

DESCRIPTION: 描述：

The getcwd() function copies the absolute pathname of the current working directory to the character array pointed to by buf. The size argument is the number of bytes available in buf
getcwd()函数复制当前工作目录的绝对路径名到由 buf 指向的字符数组。size 参数是在 buf 中可用的字节数。

NOTES: 注意事项：

There is no way to determine the maximum string length that fetcwd() may need to return. Applications should tolerate getting ERANGE and allocate a larger buffer.
没有办法确定 fetcwd()可能需要返回的最大字符串长度。应用程序容忍获得 ERANGE 并分配一个更大的缓冲区。

It is possible for getcwd() to return EACCES if, say, login puts the process into a directory without read access.
可能的是 getcwd()返回 EACCES，如果是这样，login 将进程置于没有读取访问的

目录。

The 1988 standard uses int instead of size_t for the second parameter.
1988 年的标准对第二个参数使用 int 而不是 size_t。

4.4.10 open - Opens a file

4.4.10 open - 打开文件

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(
   const char *path,
   int    oflag,
   mode_t    mode
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file’s path prefix.
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EEXIST The named file already exists.
EEXIST 指定的文件已存在。

EINTR Function was interrupted by a signal.
EINTR 函数被一个信号中断。

EISDIR Attempt to open a directory for writing or to rename a file to be a directory.
EISDIR 试图为写入打开目录，或重命名文件为目录。

EMFILE Too many file descriptors are in use by this process.
EMFILE 这个进程使用了太多文件描述符。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENFILE Too many files are currently open in the system.
ENFILE 当前在系统中打开了太多文件。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOSPC No space left on disk.
ENOSPC 磁盘上没有剩余空间。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

ENXIO No such device. This error may also occur when a device is not ready, for example, a tape drive is off-line.
ENXIO 没有此类设备。当设备未就绪时，可能也会发生此错误，例如，磁带驱动器是离线的。

EROFS Read-only file system.
EROFS 只读文件系统。

DESCRIPTION: 描述：

The open function establishes a connection between a file and a file descriptor. The file descriptor is a small integer that is used by I/O functions to reference the file. The path argument points to the pathname for the file.
open 函数建立文件和文件描述符之间的连接。文件描述符是一个小的整数，用于 I/O 函数引用该文件。path 参数指向该文件的路径名。

The oflag argument is the bitwise inclusive OR of the values of symbolic constants. The programmer must specify exactly one of the following three symbols:
oflag 参数是符号常量的值的按位逻辑或。程序员必须确切地指定以下三个符号之一：

O_RDONLY Open for reading only.
O_RDONLY 以只读方式打开。

O_WRONLY Open for writing only.
O_WRONLY 以只写方式打开。

O_RDWR Open for reading and writing.
O_RDWR 以读写方式打开。

Any combination of the following symbols may also be used.
还可以使用以下符号的任意组合。

O_APPEND Set the file offset to the end-of-file prior to each write.
O_APPEND 在每次写入之前设置文件偏移量到文件末尾。

O_CREAT If the file does not exist, allow it to be created. This flag indicates that the mode argument is present in the call to open.
O_CREAT 如果文件不存在，允许创建它。此标志表示mode参数存在于调用open。

O_EXCL This flag may be used only if O_CREAT is also set. It causes the call to open to fail if the file already exists.
O_EXCL 只有 O_CREAT 也被设置时，可以使用此标志。如果文件已经存在，它导致调用 open 失败。

O_NOCTTY If path identifies a terminal, this flag prevents that teminal from becoming the controlling terminal for thi~~9~~s process. See Chapter 8 for a description of terminal I/O.
O_NOCTTY 如果 path 识别了终端，此标志防止该终端成为此进程的控制终端。有关终端 I/O 的描述，请参阅第 8 章。

O_NONBLOCK Do no wait for the device or file to be ready or available. After the file is open, the read and write calls return immediately. If the process would be delayed in the read or write oper~~m~~ation, -1 is returned and errno is set to EAGAIN instead of blocking the caller.
O_NONBLOCK 不等待设备或文件就绪或可用。在打开文件之后，read 和 write 调用立即返回。如果该进程会在读取或写入操作时延迟，返回-1 并设置 errno 为 EAGAIN，而不是阻塞调用者。

O_TRUNC This flag should be used only on ordinary files opened for writing. It causes the file to be truncated to zero length.~~.~~
O_TRUNC 这个标志应仅用于为写入打开的普通文件。它导致文件被截断为 0 长度。

Upon successful completion, open returns a non-negative file descriptor.
成功完成之后，open 返回一个非负文件描述符。

NOTES: 注意事项：

NONE
无

4.4.11 creat - Create a new file or rewrite an existing one

4.4.11 creat - 创建新文件或重写存在的文件

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int creat(
   const    char    *path,
   mode_t   mode
);
```

STATUS CODES: 状态码：

EEXIST path already exists and O_CREAT and O_EXCL were used.
EEXIST path 已存在，并且 O_CREAT 和 O_EXCL 被使用。

EISDIR path refers to a directory and the access requested involved writing
EISDIR path 引用目录，并且请求的访问涉及写入。

ETXTBSY path refers to an executable image which is currently being executed and write access was requested
ETXTBSY path 引用当前正在执行的可执行映像，并且要求写入访问。

EFAULT path points outside your accessible address space
EFAULT path 指向你可访问的地址空间之外。

EACCES The requested access to the file is not allowed, or one of the directories in path did not allow search (execute) permission.
EACCES 不允许访问请求的文件，或 path 中的目录之一不允许搜索（执行）权限。

ENAMETOOLONG path was too long.
ENAMETOOLONG path 太长。

ENOENT A directory component in path does not exist or is a dangling symbolic link.

ENOENT path 中的目录部分不存在或是悬空的符号链接。

ENOTDIR A component used as a directory in path is not, in fact, a directory.
ENOTDIR path 中用作目录的部分实际上不是目录。

EMFILE The process already~~h~~ has the maximum number of files open.
EMFILE 该进程已经有文件打开的最大数量。

ENFILE The limit on the total number of files open on the system has been reached.
ENFILE 已达到该系统上文件打开的总数的限制。

ENOMEM Insufficient kernel memory was available.
ENOMEM 没有足够的内核内存可用。

EROFS path refers to a file on a read-only filesystem and write access was requested
EROFS path 引用了只读文件系统上的文件，并且要求写入访问。

DESCRIPTION: 描述：

creat attempts to create a file and return a file descriptor for use in read, write, etc.
creat 尝试创建文件并返回用于读、写等的文件描述符。

NOTES: 注意事项：

~~NONE~~

The routine is implemented in Cygnus newlib.
该例程是在 Cygnus newlib 中实现的。

4.4.12 umask - Sets a file creation mask.

4.4.12 umask - 设置文件创建掩码

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

mode_t umask(
   mode_t cmask
);
```

STATUS CODES: 状态码：

DESCRIPTION: 描述：

The umask() function sets the process file creation mask to cmask. The file creation mask is used during open(), creat(), mkdir(), mkfifo() calls to turn off permission bits in the mode argument. Bit positions that are set in cmask are cleared in the mode of the created file.
umask()函数设置进程文件创建掩码为 cmask。文件创建掩码在 open()、creat()、mkdir()、mkfifo()调用期间使用，以关闭 mode 参数中的权限位。在 cmask 中设置的位的位置在创建的文件的模式中被清除。

NOTES: 注意事项：

~~NONE~~
The cmask argument should have only permission bits set. All other bits should be zero.
cmask 参数应只有权限位设置。所有其他位应为 0。

In a system which supports multiple processes, the file creation mask is inherited across fork() and exec() calls. This makes it possible to alter the default permission bits of created files. RTEMS does not support multiple processes so this behavior is not possible.
在支持多进程的系统中，文件创建掩码是跨 fork()和 exec()调用继承的。这使它可能改变创建文件的默认权限位。RTEMS 不支持多进程，因此这种行为是不可能的。

4.4.13 link - Creates a link to a file

4.4.13 link - 创建指向文件的链接

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int link(
   const   char   *existing,
   const char *new
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file’s path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EEXIST The named file already exists.
EEXIST 指定的文件已存在。

EMLINK The number of links would exceed LINK_MAX.
EMLINK 链接的数量超过 LINK_MAX。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOSPC No space left on disk.
ENOSPC 磁盘上没有剩余空间。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

EPERM Operation is not permitted. Process does not have the appropriate

priviledges or permissions to perform the requested operations.
EPERM 不允许操作。进程没有适当的特权或权限来执行请求的操作。

EROFS Read-only file system.
EROFS 只读文件系统。

EXDEV Attempt to link a file to another file system.
EXDEV 试图链接文件到另一个文件系统。

DESCRIPTION: 描述：

The link() function atomically creates a new link for an existing file and increments the link count for the file.
link()函数为现有的文件原子地创建新链接，并为该文件增加链接计数。

If the link() function fails, no directories are modified.
如果 link()函数失败，则没有目录被修改。

The existing argument should not be a directory.
existing 参数不应是目录。

The caller may (or may not) need permission to access the existing file.
调用者可能（或不可能）需要权限来访问现有文件。

NOTES: 注意事项：

NONE
无

4.4.14 symlink - Creates a symbolic link to a file

4.4.14 symlink - 创建指向文件的符号连接

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int symlink(
   const   char   *topath,
   const char *frompath
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file's path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EEXIST The named file already exists.
EEXIST 指定的文件已存在。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOSPC No space left on disk.
ENOSPC 磁盘上没有剩余空间。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

EPERM Operation is not permitted. Process does not have the appropriate priviledges or permissions to perform the requested operations.
EPERM 不允许操作。进程没有适当的特权或权限来执行请求的操作。

EROFS Read-only file system.
EROFS 只读文件系统。

DESCRIPTION: 描述：

The symlink() function creates a symbolic link from the frombath to the topath. The symbolic link will be interpreted at run-time.
symlink()函数创建从 frombath 到 topath 的符号链接。符号链接将在运行时解释。

If the symlink() function fails, no directories are modified.
如果 symlink()函数失败，则没有目录被修改。

The caller may (or may not) need permission to access the existing file.
调用者可能（可不可能）需要权限来访问现有文件。

NOTES: 注意事项：

NONE
无

4.4.15 readlink - Obtain the name of a symbolic link destination

4.4.15 readlink - 获取符号链接目标的名称

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int readlink(
   const char *path,
   char    *buf,
   size_t    bufsize
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file’s path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOTDIR A component of the prefix pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，前缀路径名的一部分不是目录。

ELOOP Too many symbolic links were encountered in the pathname.
ELOOP 在路径名中遇到太多符号链接。

EINVAL The pathname does not refer to a symbolic link
EINVAL 路径名没有引用符号链接。

EFAULT An invalid pointer was passed into the readlink() routine.
EFAULT 无效指针被传递到 readlink()例程。

DESCRIPTION:　描述：

The readlink() function places the symbolic link destination into buf argument and returns the number of characters copied.
readlink()函数放置符号链接目标到 buf 参数，并返回复制的字符数。

If the symbolic link destination is longer than bufsize characters the name will be truncated.
如果符号链接目标比 bufsize 字符更长，该名称将被截断。

NOTES:　注意事项：

NONE
无

4.4.16 mkdir - Makes a directory

4.4.16 mkdir - 创建目录

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

int mkdir(
   const    char    *path,
   mode_t   mode
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file's path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EEXIST The name file already exist.
EEXIST 名称文件已存在。

EMLINK The number of links would exceed LINK_MAX
EMLINK 链接的数量超过 LINK_MAX。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOSPC No space left on disk.
ENOSPC 磁盘上没有剩余空间。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

EROFS Read-only file system.
EROFS 只读文件系统。

DESCRIPTION: 描述：

The mkdir() function creates a new directory named path. The permission bits (modified by the file creation mask) are set from mode. The owner and group IDs for the directory are set from the effective user ID and group ID.
mkdir()函数创建名为 path 的新目录。权限位（通过文件创建掩码修改）从 mode 设置。该目录的所有者和组 ID 从有效的用户 ID 和组 ID 设置。

The new directory may (or may not) contain entries for.. and .. but is otherwise empty.
新目录可能（或不可能）包含.和..的项，但另外的部分为空。

NOTES: 注意事项：

NONE
无

4.4.17 mkfifo - Makes a FIFO special file

4.4.17 mkfifo - 创建 FIFO 特殊文件

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(
   const     char     *path,
   mode_t    mode
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file’s path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EEXIST The named file already exists.
EEXIST 指定的文件已存在。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOSPC No space left on disk.
ENOSPC 磁盘上没有剩余空间。

ENOTDIR A component of the specified path was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的 path 的一部分不是目录。

EROFS Read-only file system.
EROFS 只读文件系统。

DESCRIPTION: 描述：

The mkfifo() function creates a new FIFO special file named path. The permission bits (modified by the file creation mask) are set from mode. The owner and group IDs for the FIFO are set from the efective user ID and group ID.

mkfifo()函数创建名为 path 的新 FIFO 特殊文件。权限位（通过文件创建掩码修改）从 mode 设置。该 FIFO 的所有者和组 ID 从有效的用户 ID 和组 ID 设置。

NOTES: 注意事项：

NONE
无

4.4.18 unlink - Removes a directory entry

4.4.18 unlink - 移除目录项

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int unlink(
   const   char   path
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file’s path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EBUSY The directory is in use.
EBUSY 该目录正在使用中。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified path was not a directory when a direc- tory was expected.
ENOTDIR 当目录是期望的目录时，指定的 path 的一部分不是目录。

EPERM Operation is not permitted. Process does not have the appropriate priviledges or permissions to perform the requested operations.
EPERM 不允许操作。进程没有适当的特权或权限来执行请求的操作。

EROFS Read-only file system.
EROFS 只读文件系统。

DESCRIPTION:　描述：

The unlink function removes the link named by path and decrements the link count of the file referenced by the link. When the link count goes to zero and no process has the file open, the space occupied by the file is freed and the file is no longer accessible.
unlink 函数移除 path 所指定的 link，并减少链接所引用的文件的链接计数。当链接计数变为 0 并且没有进程打开该文件，则释放该文件所占的空间，并且该文件不再可访问。

NOTES:　注意事项：

NONE
无

4.4.19 rmdir - Delete a directory

4.4.19 rmdir - 删除目录

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int rmdir(
   const char *pathname
);
```

STATUS CODES: 状态码：

EPERM The filesystem containing pathname does not support the removal of directories.
EPERM 包含 pathname 的文件系统不支持移除目录。

EFAULT pathname points ouside your accessible address space.
EFAULT pathname 指向你可访问的地址空间之外。

EACCES Write access to the directory containing pathname was not allowed for the process's effective uid, or one of the directories in pathname did not allow search (execute) permission.
EACCES 对包含 pathname 目录的写入访问不允许进程的有效 uid，或 pathname 中的目录之一不允许搜索（执行）权限。

EPERM The directory containing pathname has the stickybit (S_ISVTX) set and the process's effective uid is neither the uid of the file to be delected nor that of the directory containing it.
EPERM 包含 pathname 的目录有 stickybit (S_ISVTX)设置，进程的有效 uid 既不是要被删除的文件的 uid，也不是包含它的目录的 uid。

ENAMETOOLONG pathname was too long.
ENAMETOOLONG pathname 太长。

ENOENT A dirctory component in pathname does not exist or is a dangling symbolic link.
ENOENT pathname 中的目录部分不存在或是悬空的符号链接。

ENOTDIR pathname, or a component used as a directory in pathname, is not, in fact, a directory.
ENOTDIR pathname 或在 pathname 中用作目录的部分实际上不是目录。

ENOTEMPTY pathname contains entries other than . and .. .
ENOTEMPTY pathname 包含除了.和..的项。

EBUSY pathname is the current working directory or root directory of some process
EBUSY pathname 是当前工作目录或某些进程的根目录。

EBUSY pathname is the current directory or root directory of some process.
EBUSY pathname 是当前目录或某些进程的根目录。

ENOMEM Insufficient kernel memory was available
ENOMEM 没有足够的内核内存可用。

EROGS pathname refers to a file on a read-only filesystem.
EROGS pathname 引用了只读文件系统上的文件。

ELOOP pathname contains a reference to a circular symbolic link
ELOOP pathname 包含对循环符号链接的引用。

DESCRIPTION: 描述：

rmdir deletes a directory, which must be empty
rmdir 删除目录，该目录必须为空。

NOTES: 注意事项：

NONE
无

4.4.20 rename - Renames a file

4.4.20 rename - 重命名文件

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int rename(
  const  char  *old,
  const char *new
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file’s path prefix.
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EBUSY The directory is in use.
EBUSY 该目录正在使用中。

EEXIST The named file already exists.
EEXIST 指定的文件已存在。

EINVAL Invalid argument.
EINVAL 无效的参数。

EISDIR Attempt to open a directory for writing or to rename a file to be a directory.
EISDIR 试图为写入打开目录，或重命名文件为目录。

EMLINK The number of links would exceed LINK_MAX.
EMLINK 链接的数量超过 LINK_MAX。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does no exist.
ENOENT 文件或目录不存在。

ENOSPC No space left on disk.
ENOSPC 磁盘上没有剩余空间。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

ENOTEMPTY Attempt to delete or rename a non-empty directory.
ENOTEMPTY 试图删除或重命名非空目录。

EROFS Read-only file system
EROFS 只读文件系统。

EXDEV Attempt to link a file to another file system.
EXDEV 试图链接文件到另一个文件系统。

DESCRIPTION: 描述：

The rename() function causes the file known bo old to now be known as new.
rename()函数导致被称为 old 的文件现在被称为 new。

Ordinary files may be renamed to ordinary files, and directories may be renamed to directories; however, files cannot be converted using rename(). The new pathname may not contain a path prefix of old.
普通文件可以被重命名为普通文件，目录可以被重命名为目录，然而，不能使用 rename()转换文件，new 路径名可以不包含 old 的路径前缀。

NOTES: 注意事项：

If a file already exists by the name new, it is removed. The rename() function is atomic. If the rename() detects an error, no files are removed. This guarantees that the rename("x", "x") does not remove x.
如果一个文件已存在名称 new，移除它。rename()函数是原子的。如果 rename()检测到错误，没有文件被移除。这保证 rename("x", "x")不会移除 x。

You may not rename dot or dot-dot.
你不能重命名点或点点。

The routine is implemented in Cygnus newlib using link() and unlink().

该例程是使用 link()和 unlink()在 Cygnus newlib 中实现的。

4.4.21 stat - Gets information about a file

4.4.21 stat - 获得关于文件的信息

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(
   const char    *path,
   struct stat *buf
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file's path prefix.
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EBADF Invalid file descriptor.
EBADF 无效的文件描述符。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

DESCRIPTION: 描述：

The path argument points to a pathname for a file. Read, write, or execute permission for the file is not required, but all directories listed in path must be searchable. The stat() function obtains information about the named file and writes it to the area pointed to by buf.

path 参数指向文件的路径名。不需要文件的读、写或执行权限，但在 path 中列出的所有目录必须可搜索。stat()函数获取关于指定文件的信息，并将它写到 buf 所指向的区域。

NOTES:　注意事项：

NONE
无

4.4.22 fstat - Gets file status

4.4.22 fstat - 获得文件状态

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

int fstat(
   int    fildes,
   struct stat *buf
);
```

STATUS CODES: 状态码：

EBADF	Invalid file descriptor
EBADF	无效的文件描述符

DESCRIPTION: 描述：

The fstat() function obtains information about the file associated with fildes and writes it to the area pointed to by the buf argument.
fstat()函数获取关于与 fildes 相关联的文件的信息，并将它写到 buf 参数所指向的区域。

NOTES: 注意事项：

If the filesystem object referred to by fildes is a link, then the information returned in buf refers to the destination of that link. This is in contrast to lstat() which does not follow the link.
如果 fildes 所引用的文件系统对象是一个链接，则在 buf 中返回的信息引用该链接的目标。这与不跟随该链接的 lstat()形成对比。

4.4.23 lstat - Gets file status

4.4.23 lstat - 获得文件状态

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

int lstat(
   int    fildes,
   struct stat *buf
);
```

STATUS CODES: 状态码：

EBADF	Invalid file descriptor
EBADF	无效的文件描述符

DESCRIPTION: 描述：

The lstat() function obtains information about the file associated with fildes and writes it to the area pointed to by the buf argument.
lstat()函数获取关于与 fildes 相关联的文件的信息，并将它写到 buf 参数所指向的区域。

NOTES: 注意事项：

If the filesystem object referred to by fildes is a link, then the information returned in buf refers to the link itself. This is in contrast to fstat() which follows the link.
如果 fildes 所引用的文件系统对象是一个链接，则在 buf 中返回的信息引用该链接本身。这与跟随该链接的 fstat()形成对比。

The lstat() routine is defined by BSD 4.3 and SVR4 and not included in POSIX 1003.1b-1996.
lstat()例程由 BSD 4.3 和 SVR4 定义，不包括在 POSIX 1003.1b-1996 中。

4.4.24 access - Check permissions for a file

4.4.24 access - 检查文件权限

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int access(
   const char *pathname,
   int    mode
);
```

STATUS CODES: 状态码：

EACCES The requested access would be denied, either to the file itself or one of the directories in pathname.
EACCES 对于文件本身或 pathname 中的目录之一，请求的访问被拒绝。

EFAULT pathname points outside your accessible address space.
EFAULT pathname 指向你可访问的地址空间之外。

EINVAL Mode was incorrectly specified.
EINVAL mode 被错误地指定。

ENAMETOOLONG pathname is too long.
ENAMETOOLONG pathname 太长。

ENOENT A directory component in pathname would have been accessible but does not exist or was a dangling symbolic link.
ENOENT pathname 中的目录部分已可被访问但不存在，或它是悬空的符号链接。

ENOTDIR A component used as a directory in pathname is not, in fact, a directory.
ENOTDIR 在 pathname 中用作目录的部分实际上不是目录。

ENOMEM Insufficient kernel memory was available.
ENOMEM 没有足够的内核内存可用。

DESCRIPTION: 描述：

Access checks whether the process would be allowed to read, write or test for existence of the file (or other file system object) whose name is pathname. If pathname is a symbolic link permissions of the file referred by this symbolic link are tested.
Access 检查是否允许进程读取、写入或名为 pathname 的文件（或其他文件系统对象）的存在的测试。如果 pathname 是一个符号链接，则测试此符号链接所引用的文件的权限。

Mode is a mask consisting of one or more of R_OK, W_OK, X_OK and F_OK.
mode 是包含一个或多个 R_OK、W_OK、X_OK 和 F_OK 的掩码。

NOTES:　注意事项：

NONE
无

4.4.25 chmod - Changes file mode.

4.4.25 chmod - 更改文件模式

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod(
   const     char     *path,
   mode_t    mode
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file's path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

EPERM Operation is not permitted. Process does not have the appropriate priviledges or permissions to perform the requested operations.
EPERM 不允许操作。进程没有适当的特权或权限来执行请求的操作。

EROFS Read-only file system.
EROFS 只读文件系统。

DESCRIPTION: 描述：

Set the file permission bits, the set user ID bit, and the set group ID bit for the file named by path to mode. If the effective user ID does not match the owner of the file and the calling process does not have the appropriate privileges, chmod() returns -1 and sets errno to EPERM.
对 path 所指定的文件设置文件权限位、设置用户 ID 位和设置组 ID 位为 mode。如果有效的用户 ID 匹配该文件的所有者，并且调用进程没有适当的特权，chmod() 返回-1 并设置 errno 为 EPERM。

NOTES: 注意事项：

NONE
无

4.4.26 fchmod - Changes permissions of a file

4.4.26 fchmod - 更改文件权限

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <sys/stat.h>

int fchmod(
   int    fildes,
   mode_t mode
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file’s path prefix.
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EBADF The descriptor is not valid.
EBADF 描述符无效。

EFAULT path points outside your accessible address space.
EFAULT path 指向你可访问的地址空间之外。

EIO A low-level I/o error occurred while modifying the inode.
EIO 当修改 inode 时出现低级 I/O 错误。

ELOOP path contains a circular reference
ELOOP path 包含循环引用。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does no exist.
ENOENT 文件或目录不存在。

ENOMEM Insufficient kernel memory was avaliable.

ENOMEM 没有足够的内核内存可用。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

EPERM The effective UID does not match the owner of the file, and is not zero
EPERM 有效的 UID 不匹配该文件的所有者，并且不为 0。

EROFS Read-only file system
EROFS 只读文件系统。

DESCRIPTION: 描述：

The mode of the file given by path or referenced by filedes is changed.
path 所给的或 filedes 所引用的文件的模式被更改。

NOTES: 注意事项：

NONE
无

4.4.27 getdents - Get directory entries

4.4.27 getdents - 获得目录项

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>
#include <linux/dirent.h>
#include <linux/unistd.h>

long getdents(
   int   dd_fd,
   char *dd_buf,
   int   dd_len
);
```

STATUS CODES: 状态码：

A successful call to getdents returns ~~th~~ the number of bytes read. On end of directory, 0 is returned. When an error occurs, -1 is returned, and errno is set appropriately.
成功调用 getdents 返回读取的字节数。在目录的末尾时，返回 0。当出现错误时，回返-1，并适当地设置 errno。

EBADF Invalid file descriptor fd.
EBADF 无效的文件描述符 fd。

EFAULT Argument points outside the calling process's address space.
EFAULT 参数指向调用进程的地址空间之外。

EINVAL Result buffer is too small.
EINVAL 结果缓冲区太小。

ENOENT No such directory.
ENOENT 没有这样的目录。

ENOTDIR File descriptor does not refer to a directory.
ENOTDIR 文件描述符没有引用目录。

DESCRIPTION: 描述：

getdents reads several dirent structures from the directory pointed by fd into the memory area pointed to by dirp. The parameter count is the size of the memory area.
getdents 从 fd 所指向的目录读取几个 dirent 结构到 dirp 所指向的内存区域。参数 count 是内存区域的大小。

NOTES: 注意事项：

NONE
无

4.4.28 chown - Changes the owner and/or group of a file.

4.4.28 chown - 更改文件所有者和/或组

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <unistd.h>

int chown(
   const char *path,
   uid_t   owner,
   gid_t   group
);
```

STATUS CODES: 状态码：

EACCES Search permission is denied for a directory in a file's path prefix
EACCES 文件路径前缀中的目录的搜索权限被拒绝。

EINVAL Invalid argument
EINVAL 无效的参数

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist.
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified pathname was not a directory when a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的路径名的一部分不是目录。

EPERM Operation is not permitted. Process does not have the appropriate priviledges or permissions to perform the requested operations.
EPERM 不允许操作。进程没有适当的特权或权限来执行请求的操作。

EROFS Read-only file system.

EROFS 只读文件系统。

DESCRIPTION: 描述：

The user ID and group ID of the file named by path are set to owner and ~~path~~group, respectively. For regular files, the set group ID (S_ISGID) and set user ID (S_ISUID) bits are cleared.
由 path 指定的文件的用户 ID 和组 ID 被分别设置为 owner 和 group。对于常规文件，设置的组 ID（S_ISGID）和设置的用户 ID（S_ISUID）位被清除。

Some systems consider it a security violation to allow the owner of a file to be changed, If users are billed for disk space usage, loaning a file to another user could result in incorrect billing. The chown() function may be restricted to privileged users for some or all files. The group ID can still be changed to one of the supplementary group IDs.
有些系统认为它允许更改文件所有者违反安全。如果用户对磁盘空间使用记账，将文件借给另一个用户可能导致不正确的记账。对于一些或所有文件，chown() 函数可能被限制于特权用户。组 ID 还可以被更改为一个补充组 ID。

NOTES: 注意事项：

This function may be restricted for some file. The pathconf function can be used to test the _PC_CHOWN_RESTRICTED flag.
对某些文件，此函数可能被限制。pathconf 函数可以用于测试 _PC_CHOWN_RESTRICTED 标志。

4.4.29 utime - Change access and/or modification times of an inode

4.4.29 utime - 更改 inode 的访问和/或修改时间

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>

int utime(
   const char    *filename,
   struct utimbuf *buf
);
```

STATUS CODES: 状态码：

EACCES Permission to write the file is denied
EACCES 写入该文件的权限被拒绝。

ENOENT Filename does not exist
ENOENT filename 不存在。

DESCRIPTION: 描述：

Utime changes the access and modification times of the inode specified by filename to the actime and modtime fields of buf respectively. If buf is NULL, then the access and modification times of the file are set to the current time.
Utime 分别更改 filename 所指定的 inode 的访问和修改时间到 buf 的 actime 和 modtime 字段。如果 buf 为 NULL，则该文件的访问和修改时间被设置为当前时间。

NOTES: 注意事项：

NONE
无

4.4.30 ftruncate - truncate a file to a specified length

4.4.30 ftruncate - 截断文件到指定的长度

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int ftrunctate(
   int    fd,
   size_t length
);
```

STATUS CODES: 状态码：

ENOTDIR A component of the path prefix is not a directory.
ENOTDIR 路径前缀的一部分不是目录。

EINVAL The pathname contains a character with the high-order bit set.
EINVAL 路径名包含具有高位的字符。

ENAMETOOLONG A component of a pathname exceeded 255 characters, or an entire path name exceeded 1023 characters.
ENAMETOOLONG 路径名的一部分超过了 255 个字符，或全部路径名超过了 1023 个字符。

ENOENT The named file does not exist.
ENOENT 指定的文件不存在。

EACCES The named file is not writable by the user.
EACCES 指定的文件通过用户不可写。

EACCES Search permission is denied for a component of the path prefix.
EACCES 对于路径前缀的一部分，搜索权限被拒绝。

ELOOP Too many symbolic links were encountered in translating the pathname
ELOOP 在转换路径名时遇到太多符号链接。

EISDIR The named file is a directory.
EISDIR 指定的文件是目录。

EROFS The named file resides on a read-only file system
EROFS 指定的文件位于只读文件系统。

ETXTBSY The file is a pure procedure (shared text) file that is being executed
ETXTBSY 该文件是一个正在被执行的纯程序（共享的文本）文件。

EIO An I/O error occurred updating the inode.
EIO 更新 inode 时出现 I/O 错误。

EFAULT Path points outside the process's allocated address space.
EFAULT Path 指向该进程分配的地址空间之外。

EBADF The fd is not a valid descriptor.
EBADF fd 不是有效的描述符。

DESCRIPTION: 描述：

truncate() causes the file named by path or referenced by fd to be truncated to at most length bytes in size. If the file previously was larger than this size, the extra data is lost. With ftruncate(), the file must be open for writing.
truncate()导致 path 所指定名称的或 fd 所引用的文件被截断为最多 length 字节的大小。如果该文件先前大于此大小，额外的数据将丢失。对于 ftruncate()，该文件必须为写入打开。

NOTES: 注意事项：

NONE
无

4.4.31 truncate - truncate a file to a specified length

4.4.31 truncate - 截断文件到指定的权限

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int trunctate(
   const char *path,
   size_t    length
);
```

STATUS CODES: 状态码：

ENOTDIR A component of the path prefix is not a directory.
ENOTDIR 路径前缀的一部分不是目录。

EINVAL The pathname contains a character with the high-order bit set.
EINVAL 路径名包含具有高位的字符。

ENAMETOOLONG A component of a pathname exceeded 255 characters, or an entire path name exceeded 1023 characters.
ENAMETOOLONG 路径名的一部分超过了 255 个字符，或全部路径名超过了 1023 个字符。

ENOENT The named file does not exist.
ENOENT 指定的文件不存在。

EACCES The named file is not writable by the user.
EACCES 指定的文件通过用户不可写。

EACCES Search permission is denied for a component of the path prefix.
EACCES 对于路径前缀的一部分，搜索权限被拒绝。

ELOOP Too many symbolic links were encountered in translating the pathname
ELOOP 在转换路径名时遇到太多符号链接。

EISDIR The named file is a directory.
EISDIR 指定的文件是目录。

EROFS The named file resides on a read-only file system
EROFS 指定的文件位于只读文件系统。

ETXTBSY The file is a pure procedure (shared text) file that is being executed
ETXTBSY 该文件是一个正在被执行的纯程序（共享的文本）文件。

EIO An I/O error occurred updating the inode.
EIO 更新 inode 时出现 I/O 错误。

EFAULT Path points outside the process's allocated address space.
EFAULT Path 指向该进程分配的地址空间之外。

EBADF The fd is not a valid descriptor.
EBADF fd 不是有效的描述符。

DESCRIPTION: 描述：

truncate() causes the file named by path or referenced by fd to be truncated to at most length bytes in size. If the file previously was larger than this size, the extra data is lost. With ftruncate(), the file must be open for writing.
truncate()导致 path 所指定名称的或 fd 所引用的文件被截断为最多 length 字节的大小。如果该文件先前大于此大小，额外的数据将丢失。对于 ftruncate()，该文件必须为写入打开。

NOTES: 注意事项：

NONE
无

4.4.32 pathconf - Gets configuration values for files

4.4.32 pathconf - 获得文件的配置值

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int pathconf(
   const char *path,
   int    name
);
```

STATUS CODES: 状态码：

EINVAL Invalid argument
EINVAL 无效的参数。

EACCES Permission to write the file is denied
EACCES 写入该文件的权限被拒绝。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified path was not a directory whan a directory was expected.
ENOTDIR 当目录是期望的目录时，指定的 path 的一部分不是目录。

DESCRIPTION: 描述：

pathconf() gets a value for the configuration option name for the open file descriptor filedes.
pathconf()获得打开文件描述符 filedes 的配置选项 name 的值。

The possible values for name are:

可能的 name 值有：

_PC_LINK_MAX	returns the maximum number of links to the file. If filedes or path refer to a directory, then the value applies to the whole directory. The corresponding macro is _POSIX_LINK_MAX.
_PC_LINK_MAX	返回指向该文件的链接的最大数量。如果 filedes 或 path 引用了一个目录，则该值应用于整个目录。对应的宏是_POSIX_LINK_MAX。
_PC_MAX_CANON	returns the maximum length of a formatted input line, where filedes or path must refer to a terminal. The corresponding macro is _POSIX_MAX_CANON.
_PC_MAX_CANON	返回格式化的输入行的最大长度，其中，filedes 或 path 必须引用到终端。对应的宏是_POSIX_MAX_CANON。
_PC_MAX_INPUT	returns the maximum length of an input line, where filedes or path must refer to a terminal. The corresponding macro is _POSIX_MAX_INPUT.
_PC_MAX_INPUT	返回输入行的最大长度，其中，filedes 或 path 必须引用到终端。对应的宏是_POSIX_MAX_INPUT。
_PC_NAME_MAX	returns the maximum length of a filename in the directory path or filedes. The process is allowed to create. The corresponding macro is _POSIX_NAME_MAX.
_PC_NAME_MAX	返回 filedes 或 path 中的文件名的最大长度，允许该进程创建。对应的宏是_POSIX_NAME_MAX。
_PC_PATH_MAX	returns the maximum length of a relative pathname when path or filedes is the current working directory. The corresponding macro is _POSIX_PATH_MAX.
_PC_PATH_MAX	当 path 或 filedes 是当前工作目录时，返回相对路径名的最大长度。对应的宏是_POSIX_PATH_MAX。
_PC_PIPE_BUF	returns the size of the pipe buffer, where filedes must refer to a pipe or FIFO and path must refer to a FIFO. The corresponding macro is _POSIX_PIPE_BUF.
_PC_PIPE_BUF	返回管道缓冲区的大小，其中，filedes 必须引用到管道或 FIFO，path 必须引用到 FIFO。对应的宏是_POSIX_PIPE_BUF。

_PC_CHOWN_RESTRICTED

returns nonzero if the chown(2) call may not be used on this file. If filedes or path refer to a directory, then this applies to all files in that directory. The corresponding macro is _POSIX_CHOWN_RESTRICTED.

如果 chown(2)调用不能用于此文件，返回非零值。如果 filedes 或 path 引用到目录，则这应用于该目录中的所有文件。对应的宏是_POSIX_CHOWN_RESTRICTED。

NOTES: 注意事项：

Files with name lengths longer than the value returned for name equal _PC_NAME_MAX may exist in the given directory.

具有等于_PC_NAME_MAX 的、超过 name 所返回的值的名称长度的文件可能存在于给定的目录中。

4.4.33 fpathconf - Gets configuration values for files

4.4.33 fpathconf - 获得文件的配置值

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int fpathconf(
   int filedes,
   int name
);
```

STATUS CODES: 状态码：

EINVAL Invalid argument
EINVAL 无效的参数。

EACCES Permission to write the file is denied
EACCES 写入该文件的权限被拒绝。

ENAMETOOLONG Length of a filename string exceeds PATH_MAX and _POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 文件名字符串的长度超过 PATH_MAX，并且_POSIX_NO_TRUNC 生效。

ENOENT A file or directory does not exist
ENOENT 文件或目录不存在。

ENOTDIR A component of the specified path was not a directory whan a di- rectory was expected.
ENOTDIR 当目录是期望的目录时，指定的 path 的一部分不是目录。

DESCRIPTION: 描述：

pathconf() gets a value for the configuration option name for the open file descriptor filedes.
pathconf()获得打开文件描述符 filedes 的配置选项 name 的值。

The possible values for name are:

可能的 name 值有：

_PC_LINK_MAX	returns the maximum number of links to the file. If filedes or path refer to a directory, then the value applies to the whole directory. The corresponding macro is _POSIX_LINK_MAX.
_PC_LINK_MAX	返回指向该文件的链接的最大数量。如果 filedes 或 path 引用了一个目录，则该值应用于整个目录。对应的宏是_POSIX_LINK_MAX。
_PC_MAX_CANON	returns the maximum length of a formatted input line, where filedes or path must refer to a terminal. The corresponding macro is _POSIX_MAX_CANON.
_PC_MAX_CANON	返回格式化的输入行的最大长度，其中，filedes 或 path 必须引用到终端。对应的宏是_POSIX_MAX_CANON。
_PC_MAX_INPUT	returns the maximum length of an input line, where filedes or path must refer to a terminal. The corresponding macro is _POSIX_MAX_INPUT.
_PC_MAX_INPUT	返回输入行的最大长度，其中，filedes 或 path 必须引用到终端。对应的宏是_POSIX_MAX_INPUT。
_PC_NAME_MAX	returns the maximum length of a filename in the directory path or filedes. The process is allowed to create. The corresponding macro is _POSIX_NAME_MAX.
_PC_NAME_MAX	返回 filedes 或 path 中的文件名的最大长度，允许该进程创建。对应的宏是_POSIX_NAME_MAX。
_PC_PATH_MAX	returns the maximum length of a relative pathname when path or filedes is the current working directory. The corresponding macro is _POSIX_PATH_MAX.
_PC_PATH_MAX	当 path 或 filedes 是当前工作目录时，返回相对路径名的最大长度。对应的宏是_POSIX_PATH_MAX。
_PC_PIPE_BUF	returns the size of the pipe buffer, where filedes must refer to a pipe or FIFO and path must refer to a FIFO. The corresponding macro is _POSIX_PIPE_BUF.
_PC_PIPE_BUF	返回管道缓冲区的大小，其中，filedes 必须引用到管道或 FIFO，path 必须引用到 FIFO。对应的宏是_POSIX_PIPE_BUF。

_PC_CHOWN_RESTRICTED

returns nonzero if the chown(2) call may not be used on this file. If filedes or path refer to a directory, then this applies to all files in that directory. The corresponding macro is _POSIX_CHOWN_RESTRICTED.

如果 chown(2)调用不能用于此文件，返回非零值。如果 filedes 或 path 引用到目录，则这应用于该目录中的所有文件。对应的宏是_POSIX_CHOWN_RESTRICTED。

NOTES: 注意事项：

NONE
无

4.4.34 mknod - create a directory

4.4.34 mknod - 创建目录

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

long mknod(
   const   char   *pathname,
   mode_t   mode,
   dev_t   dev
);
```

STATUS CODES: 状态码：

mknod returns zero on success, or -1 if an error occurred (in which case, errno is set appropriately).
mknod 在成功时返回 0，如果出现错误返回-1（在这种情况下，errno 被适当地设置）。

ENAMETOOLONG pathname was too long.
ENAMETOOLONG pathname 太长。

ENOENT A directory component in pathname does not exist or is a dangling symbolic link.
ENOENT pathname 中的目录部分不存在或是悬空的符号链接。

ENOTDIR A component used in the directory pathname is not, in fact, a directory.
ENOTDIR 在目录 pathname 中使用的一部分实际上不是目录。

ENOMEM Insufficient kernel memory was available
ENOMEM 没有足够的内核内存可用。

EROFS pathname refers to a file on a read-only filesystem.
EROGS pathname 引用了只读文件系统上的文件。

ELOOP pathname contains a reference to a circular symbolic link, ie a symbolic link whose expansion contains a reference to itself.
ELOOP pathname 包含对循环符号链接的引用，即扩展包含对自身的引用的符号链接。

ENOSPC The device containing pathname has no room for the new node.
ENOSPC 包含 pathname 的设备没有给新节点的空间。

DESCRIPTION: 描述：

mknod attempts to create a filesystem node (file, device special file or named pipe) named pathname, specified by mode and dev.
mknod 尝试创建 mode 和 dev 所指定的名为 pathname 的文件系统节点（文件、设备特殊文件或命名管道）。

mode specifies both the permissions to use and the type of node to be created.
mode 指定使用权限和要创建的节点的类型。

It should be a combination (using bitwise OR) of one of the file types listed below and the permissions for the new node.
它应是由下面列出的文件类型之一和新节点的权限组成（使用按位或）。

The permissions are modified by the process's umask in the usual way: the permissions of the created node are (mode & ~umask).
权限由该进程的 umask 以通常的方式修改：创建的节点的权限是(mode & ~umask)。

The file type should be one of S_IFREG, S_IFCHR, S_IFBLK and S_IFIFO to specify a normal file (which will be created empty), character special file, block special file or FIFO (named pipe), respectively, or zero, which will create a normal file.
文件类型应是 S_IFREG、S_IFCHR、S_IFBLK 和 S_IFIFO 之一，以分别指定常规文件（这将被创建为空）、字符特殊文件、块特殊文件或 FIFO（命名管道），或是 0，这将创建常规文件。

If the file type is S_IFCHR or S_IFBLK then dev specifies the major and minor numbers of the newly created device special file; otherwise it is ignored.
如果文件类型是 S_IFCHR 或 S_IFBLK，则 dev 指定新创建的设备特殊文件的主、次设备号；否则它被忽略。

The newly created node will be owned by the effective uid of the process. If the directory containing the node has the set group id bit set, or if the filesystem is mounted with BSD group semantics, the new node will inherit the group ownership from its parent directory; otherwise it will be owned by the effective gid of the process.
新创建的节点将由该进程的有效 uid 拥有。如果包含该节点的目录有设置的组 id 位设置，或使用 BSD 组语义挂载该文件系统，则从其父目录继承组所有权；否则它将由该进程的有效 gid 拥有。

NOTES: 注意事项：

NONE
无

5 Input and Output Primitives Manager

5 输入和输出原始管理器

5.1 Introduction

5.1 简介

The input and output primitives manager is ...
输入和输出原始管理器是……

The directives provided by the input and output primitives manager are:
由输入和输出原始管理器提供的指令有：

- pipe - Create an Inter-Process Channel
- dup - Duplicates an open file descriptor
- dup2 - Duplicates an open file descriptor
- close - Closes a file
- read - Reads from a file
- write - Writes to a file
- fcntl - Manipulates an open file descriptor
- lseek - Reposition read/write file offset
- fsync - Synchronize file complete in-core state with that on disk
- fdatasync - Synchronize file in-core data with that on disk
- sync - Schedule file system updates
- mount - Mount a file system
- unmount - Unmount file systems
- readv - Vectored read from a file
- writev - Vectored write to a file
- aio_read - Asynchronous Read
- aio_write - Asynchronous Write
- lio_listio - List Directed I/O
- aio_error - Retrieve Error Status of Asynchronous I/O Operation
- aio_return - Retrieve Return Status Asynchronous I/O Operation
- aio_cancel - Cancel Asynchronous I/O Request
- aio_suspend - Wait for Asynchronous I/O Request
- aio_fsync - Asynchronous File Synchronization

- pipe – 创建进程间通道

- dup – 复制打开文件描述符
- dup2 -复制打开文件描述符
- close – 关闭文件
- read – 从文件读取
- write – 写入到文件
- fcntl – 操纵打开文件描述符
- lseek - 重定位读/写文件偏移量
- fsync – 将文件的完整核心内状态与磁盘上的状态同步
- fdatasync – 将文件的核心内数据与磁盘上的数据同步
- sync – 安排文件系统更新
- mount – 挂载文件系统
- unmount – 卸载文件系统
- readv – 从文件定向读取
- writev – 定向写入到文件
- aio_read – 异步读取
- aio_write – 异步写入
- lio_listio - I/O 所指向的列表
- aio_error – 检索异步 I/O 操作的错误状态
- aio_return - 检索异步 I/O 操作的返回状态
- aio_cancel – 取消异步 I/O 请求
- aio_suspend – 等待异步 I/O 请求
- aio_fsync – 异步文件同步化

5.2 Background

5.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

5.3 Operations

5.3 操作

There is currently no text in this section.
当前在本节中没有文本。

5.4 Directives

5.4 指令

This section details the input and output primitives manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了输入和输出原始管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

5.4.1 pipe - Create an Inter-Process Channel

5.4.1 pipe - 创建进程间通道

CALLING SEQUENCE: 调用序列：

```
int pipe(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.2 dup - Duplicates an open file descriptor

5.4.2 dup - 复制打开文件描述符

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int dup(
   int fildes
);
```

STATUS CODES: 状态码：

EBADF Invalid file descriptor.
EBADF 无效的文件描述符。

EINTR Function was interrupted by a signal.
EINTR 函数被一个信号中断。

EMFILE The process already has the maximum number of file descriptors open and tried to open a new one.
EMFILE 进程已打开最大数量的文件描述符，并试图打开一个新的文件描述符。

DESCRIPTION: 描述：

The dup function returns the lowest numbered available file descriptor. This new descriptor refers to the same open file as the original descriptor and shares any locks.
dup 函数返回编号最小的可用文件描述符。这个新的描述符引用与原始描述符相同的打开文件，并共享所有锁。

NOTES: 注意事项：

NONE
无

5.4.3 dup2 - Duplicates an open file descriptor

5.4.3 dup2 - 复制打开文件描述符

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int dup2(
   int fildes,
   int fildes2
);
```

STATUS CODES: 状态码：

EBADF Invalid file descriptor.
EBADF 无效的文件描述符。

EINTR Function was interrupted by a signal.
EINTR 函数被一个信号中断。

EMFILE The process already has the maximum number of file descriptors open and tried to open a new one.
EMFILE 进程已打开最大数量的文件描述符，并试图打开一个新的文件描述符。

DESCRIPTION: 描述：

dup2 creates a copy of the file descriptor oldfd.
dup2 创建文件描述符 oldfd 的副本。

The old and new descriptors may be used interchangeably. They share locks, file position pointers and flags; for example, if the file position is modified by using lseek on one of the descriptors, the position is also changed for the other.
旧的和新的描述符可以互换使用。它们共享锁、文件位置指针和标志；例如，如果文件位置通过使用在一个描述符上的 lseek 修改，对于另一个描述符，该位置也被更改。

NOTES: 注意事项：

NONE

无

5.4.4 close - Closes a file

5.4.4 close - 关闭文件

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int close(
   int fildes
);
```

STATUS CODES: 状态码：

EBADF Invalid file descriptor
EBADF 无效的文件描述符

EINTR Function was interrupted by a signal.
EINTR 函数被一个信号中断。

DESCRIPTION: 描述：

The close() function deallocates the file descriptor named by fildes and makes it available for reuse. All outstanding record locks owned by this process for the file are unlocked.
close()函数释放由 fildes 指定的文件描述符，使它可供重复使用。由此进程所拥有的该文件的所有未解决的记录锁被解锁。

NOTES: 注意事项：

A signal can interrupt the close() function. In that case, close() returns -1 with errno set to EINTR. The file may or may not be closed.
信号可以中断 close()函数。在这种情况下，close()返回-1，errno 设置为 EINTR。该文件可能或不可能被关闭。

5.4.5 read - Reads from a file

5.4.5 read - 从文件读取

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int read(
   int    fildes,
   void    *buf,
   unsigned int    nbyte
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1，并设置 errno 为下列值之一：

EAGAIN The O_NONBLOCK flag is set for a file descriptor and the process would be delayed in the I/O operation.
EAGAIN O_NONBLOCK 标志为文件描述符设置，该进程将在 I/O 操作时被延时。

EBADF Invalid file descriptor
EBADF 无效的文件描述符

EINTR Function was interrupted by a signal.
EINTR 函数被一个信号中断。

EIO Input or output error
EIO 输入或输出错误。

EINVAL Bad buffer pointer
EINVAL 坏的缓冲区指针。

DESCRIPTION: 描述：

The read() function reads nbyte bytes from the file associated with fildes into the buffer pointed to by buf.
read()函数从与 fildes 相关联的文件读取 nbyte 字节到 buf 所指向的缓冲区。

The read() function returns the number of bytes actually read and placed in the buffer. This will be less than nbyte if:
read()函数返回实际读取并放置在缓冲区的字节数。如果出现以下情况，这将会少于 nbyte：

- The number of bytes left in the file is less than nbyte.
- The read() request was interrupted by a signal.
- The file is a pipe or FIFO or special file with less than nbytes immediately available for reading.

- 该文件中剩余的字节数少于 nbyte。
- read()请求被一个信号中断。
- 该文件是管道、FIFO 或特殊文件，具有少于 nbytes 的对读取立即可用的字节数。

When attempting to read from any empty pipe or FIFO:
当试图从任何空的管道或 FIFO 读取时：

- If no process has the pipe open for writing, zero is returned to indicate end-of-file.
- If some process has the pipe open for writing and O_NONBLOCK is set, -1 is returned and errno is set to EAGAIN.
- If some process has the pipe open for writing and O_NONBLOCK is clear, read() waits for some data to be written or the pipe to be closed.

- 如果没有进程有为写入打开的管道，返回 0 以表示文件的末尾。
- 如果某些进程有为写入打开的管道，并设置了 O_NONBLOCK，返回-1 并设置 errno 为 EAGAIN。
- 如果某些进程有为写入打开的管道，并清除了 O_NONBLOCK，read()等待某些数据被写入或等待管道被关闭。

When attempting to read from a file other than a pipe or FIFO and no data is available.
当试图从除了管道或 FIFO 的文件读取，并且没有数据可用时：

- If O_NONBLOCK is set, -1 is returned and errno is set to EAGAIN.
- If O_NONBLOCK is clear, read() waits for some data to become available.

- The O_NONBLOCK flag is ignored if data is available.

- 如果设置了 O_NONBLOCK，返回-1 并设置 errno 为 EAGAIN。
- 如果清除了 O_NONBLOCK，read()等待某些数据变得可用。
- 如果数据是可用的，忽略 O_NONBLOCK 标志。

NOTES:　注意事项：

NONE
无

5.4.6 write - Writes to a file

5.4.6 write - 写入到文件

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

int write(
    int    fildes,
    const void    *buf,
    unsigned int    nbytes
);
```

STATUS CODES: 状态码：

EAGAIN The O_NONBLOCK flag is set for a file descriptor and the process would be delayed in the I/O operation.
EAGAIN O_NONBLOCK 标志为文件描述符设置，该进程将在 I/O 操作时被延时。

EBADF Invalid file descriptor
EBADF 无效的文件描述符

EFBIG An attempt was made to write to a file that exceeds the maximum file size
EFBIG 试图写入到超过最大文件大小的文件。

EINTR The function was interrupted by a signal.
EINTR 函数被一个信号中断。

EIO Input or output error.
EIO 输入或输出错误。

ENOSPC No space left on disk.
ENOSPC 磁盘上没有剩余空间。

EPIPE Attempt to write to a pipe or FIFO with no reader.
EPIPE 试图写入到没有读取者的管道或 FIFO。

EINVAL Bad buffer pointer
EINVAL 坏的缓冲区指针。

DESCRIPTION:　描述：

The write() function writes nbyte from the array pointed to by buf into the file associated with fildes.
write()函数从 buf 所指向的数组写入 nbyte 到与 fildes 相关联的文件。

If nybte is zero and the file is a regular file, the write() function returns zero and has no other effect. If nbyte is zero and the file is a special file, the results are not portable.
如果 nybte 为 0 并且该文件是常规文件，write()函数返回 0，并且没有其他影响。如果 nbyte 为 0 并且该文件是特殊文件，结果是不可移植。

The write() function returns the number of bytes written. This number will be less than nbytes if there is an error. It will never be greater than nbytes.
write()函数返回写入的字节数。如果有错误，这一数字将少于 nbytes。它将永远不会大于 nbytes。

NOTES:　注意事项：

NONE
无

5.4.7 fcntl - Manipulates an open file descriptor

5.4.7 fcntl - 操纵打开文件描述符

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

int    fcntl(
   int    fildes,
   int cmd
);
```

STATUS CODES: 状态码：

EACCESS Search permission is denied for a direcotry in a file’s path prefix.
EACCESS 对文件的路径前缀中的目录的搜索权限被拒绝。

EAGAIN The O_NONBLOCK flag is set for a file descriptor and the process would be delayed in the I/O operation.
EAGAIN O_NONBLOCK 标志为文件描述符设置，该进程将在 I/O 操作时被延时。

EBADF Invalid file descriptor
EBADF 无效的文件描述符

EDEADLK An fcntl with function F_SETLKW would cause a deadlock.
EDEADLK 具有函数 F_SETLKW 的 fcntl 将导致死锁。

EINTR The function was interrupted by a signal.
EINTR 函数被一个信号中断。

EINVAL Invalid argument
EINVAL 无效的参数。

EMFILE Too many file descriptor or in use by the process.
EMFILE 太多的文件描述符或由该进程使用中。

ENOLCK No locks available

ENOLCK 没有可用的锁。

DESCRIPTION: 描述：

fcntl() performs one of various miscellaneous operations on fd. The operation in question is determined by cmd:
fcntl()执行 fd 上的各种杂项操作之一。正在讨论的问题由 cmd 确定：

F_DUPFD	Makes arg be a copy of fd, closing fd first if necessary. The same functionality can be more easily achieved by using dup2(). The old and new descriptors may be used interchangeably. They share locks, file position pointers and flags; for example, if the file position is modified by using lseek() on one of the descriptors, the position is also changed for the other. The two descriptors do not share the close-on-exec flag, however. The close-on-exec flag of the copy is off, meaning that it will be closed on exec. On success, the new descriptor is returned.
F_DUPFD	使 arg 成为 fd 的副本，如果有必要，首先关闭 fd。 相同的功能可以通过使用 dup2()更容易地实现。旧的和新的描述符可以互换使用。它们共享锁、文件位置指针和标志；例如，如果文件位置通过使用在一个描述符上的 lseek()修改，对于另一个描述符，该位置也被更改。 然而，两个描述符不共享执行时关闭的标志。副本的执行时关闭的标志是关闭的，意味着它将在执行时被关闭。 在成功时，返回新描述符。
F_GETFD	Read the close-on-exec flag. If the low-order bit is 0, the file will remain open across exec, otherwise it will be closed.
F_GETFD	读取执行时关闭的标志。如果低位为 0，该文件将跨执行保持打开，否则它将被关闭。
F_SETFD	Set the close-on-exec flag to the value specified by arg (only the least significant bit is used).
F_SETFD	设置执行时关闭的标志为 arg（只使用最低有效位）所指定的值。
F_GETFL	Read the descriptor’s flags (all flags (as set by open()) are

	returned).
F_GETFL	读取描述符的标志（返回所有标志（如 open()所设置的））。
F_SETFL	Set the descriptor’s flags to the value specified by arg. Only O_APPEND and O_NONBLOCK may be set. The flags are shared between copies (made with dup() etc.) of the same file descriptor. The flags and their semantics are described in open().
F_SETFL	设置描述符的标志为 arg 所指定的值。只有 O_APPEND 和 O_NONBLOCK 可以被设置。 标志在相同文件描述符的副本（使用 dup()等产生）之间共享。 在 open()中描述标志和它们的语义。
F_GETLK, F_SETLK and F_SETLKW	
	Manage discretionary file locks. The third argument arg is a pointer to a struct flock (that may be overwritten by this call).
F_GETLK、F_SETLK 和 F_SETLKW	
	管理任意文件的锁。第三个参数 arg 是指向 struct flock 的指针（可能被此调用覆盖）。
F_GETLK	Return the flock structure that prevents us from obtaining the lock, or set the l_type field of the lock to F_UNLCK if there is no obstruction.
F_GETLK	返回 flock 结构，阻止我们获取锁，或如果没有阻档，设置锁的 l_type 字段为 F_UNLCK。
F_SETLK	The lock is set (when l_type is F_RDLCK or F_WRLCK) or cleared (when it is F_UNLCK). If lock is held by someone else, this call returns -1 and sets errno to EACCES or EAGAIN.
F_SETLK	锁被设置（当 l_type 是 F_RDLCK 或 F_WRLCK 时）或被清除（当它是 F_UNLCK 时）。如果由其他人持有锁，此调用返回 -1 并设置 errno 为 EACCES 或 EAGAIN。
F_SETLKW	Like F_SETLK, but instead of returning an error we wait for the lock to be released.
F_SETLKW	像 F_SETLK，但不返回错误，我们等待锁被释放。

F_GETOWN	Get the process ID (or process group) of the owner of a socket. Process groups are returned as negative values.
F_GETOWN	获得套接字的所有者的进程 ID（或进程组）。进程组作为负数值返回。
F_SETOWN	Set the process or process group that owns a socket. For these commands, ownership means receiving SIGIO or SIGURG signals. Process groups are specified using negative values.
F_SETOWN	设置拥有套接字的进程或进程组。 对于这些命令，所有权意味着接收 SIGIO 或 SIGURG 信号。 使用负数值指定进程组。

NOTES:　注意事项：

The errors returned by dup2 are different from those returned by F_DUPFD.
由 dup2 返回的错误不同于由 F_DUPFD 返回的那些错误。

5.4.8 lseek - Reposition read/write file offset

5.4.8 lseek - 重定位读/写文件偏移量

CALLING SEQUENCE: 调用序列：

```
#include <sys/types.h>
#include <unistd.h>

int lseek(
   int fildes,
   off_t offset,
   int      whence
);
```

STATUS CODES: 状态码：

EBADF fildes is not an open file descriptor.
EBADF fildes 不是打开文件描述符。

ESPIPE fildes is associated with a pipe, socket or FIFO.
ESPIPE fildes 与管道、套接字或 FIFO 相关联。

EINVAL whence is not a proper value.
EINVAL whence 不是适当的值。

DESCRIPTION: 描述：

The lseek function repositions the offset of the file descriptor fildes to the argument offset according to the directive whence. The argument fildes must be an open file descriptor. Lseek repositions the file pointer fildes as follows:
lseek 函数根据指令 whence，重新定位文件描述符 fildes 的偏移量到参数 offset。参数 fildes 必须是打开文件描述符。lseek 重新定位文件指针 fildes 如下所示：

- If whence is SEEK_SET, the offset is set to offset bytes.
- If whence is SEEK_CUR, the offset is set to its current location plus offset bytes.
- If whence is SEEK_END, the offset is set to the size of the file plus offset bytes.

- 如果 whence 是 SEEK_SET，偏移量被设置为 offset 字节。
- 如果 whence 是 SEEK_CUR，偏移量被设置为其当前位置加上偏移量字节。

- 如果 whence 是 SEEK_END，偏移量被设置为文件大小加上偏移量字节。

The lseek function allows the file offset to be set beyond the end of the existing end-of-file of the file. If data is later written at this point, subsequent reads of the data in the gap return bytes of zeros (until data is actually written into the gap).
lseek 函数允许文件偏移量被设置为超出文件的现有文件结束符。如果随后在此点上写入数据，随后在空白处的数据读取返回 0 字节（直到数据被实际写入到空白处）。

Some devices are incapable of seeking. The value of the pointer associated with such a device is undefined.
某些设备不能寻找。与类似设备这样的相关联的指针的值未定义。

NOTES:　注意事项：

NONE
无

5.4.9 fsync - Synchronize file complete in-core state with that on disk

5.4.9 fsync - 将文件的完整核心内状态与磁盘上的状态同步

CALLING SEQUENCE: 调用序列：

```
int fsync(
   int fd
);
```

STATUS CODES: 状态码：

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.
成功时返回 0。出现错误时返回-1，并适当地设置 errno。

EBADF fd is not a valid descriptor open for writing
EBADF fd 不是有效的为写入打开的描述符。

EINVAL fd is bound to a special file which does not support ~~support~~ synchronization
EINVAL fd 被限制为不支持同步的特殊文件。

EROFS fd is bound to a special file which does not support ~~support~~ synchronization
EROFS fd 被限制为不支持同步的特殊文件。

EIO An error occurred during synchronization
EIO 在同步期间出现错误。

DESCRIPTION: 描述：

fsync copies all in-core parts of a file to disk.
fsync 复制文件的所有核心内部分到磁盘。

NOTES: 注意事项：

NONE
无

5.4.10 fdatasync - Synchronize file in-core data with that on disk

5.4.10 fdatasync - 将文件的核心内数据与磁盘上的数据同步

CALLING SEQUENCE: 调用序列：

```
int fdatasync(
   int fd
);
```

STATUS CODES: 状态码：

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.
成功时返回 0。出现错误时返回-1，并适当地设置 errno。

EBADF fd is not a valid descriptor open for writing.
EBADF fd 不是有效的为写入打开的描述符。

EINVAL fd is bound to a special file which does not support synchronization.
EINVAL fd 被限制为不支持同步的特殊文件。

EIO An error occurred during synchronization.
EIO 在同步期间出现错误。

EROFS fd is bound to a special file which does not support synchronization.
EROFS fd 被限制为不支持同步的特殊文件。

DESCRIPTION: 描述：

fdatasync flushes all data buffers of a file to disk (before the system call returns). It resembles fsync but is not required to update the metadata such as access time.
fdatasyn 将文件的所有数据缓冲区都刷新到磁盘（在系统调用返回之前）。它类似于 fsync，但不需要更新元数据，如访问时间。

Applications that access databases or log files often write a tiny data fragment (e.g., one line in a log file) and then call fsync immediately in order to ensure that the written data is physically stored on the harddisk. Unfortunately, fsync will always initiate two write operations: one for the newly written data and another one in order to update the modification time stored in the inode. If the modification time is not a part of the transaction concept fdatasync can be used to avoid unnecessary

inode disk write operations.
访问数据库或日志文件的应用程序经常写入一个小小的数据片段（如日志文件中的一行），然后立即调用 fsync 以确保写入的数据物理地存储在硬盘上。不幸的是，fsync 将总是启动两个写入操作：一个为了新写入的数据，另一个为了更新在 inode 中存储的修改时间。如果修改时间不是事务概念的一部分，fdatasync 可以用于避免不必要的 inode 磁盘写入操作。

NOTES:　注意事项：

NONE
无

5.4.11 sync - Schedule file system updates

5.4.11 sync - 安排文件系统更新

CALLING SEQUENCE: 调用序列：

```
void sync(void);
```

STATUS CODES: 状态码：

NONE
无

DESCRIPTION: 描述：

The sync service causes all information in memory that updates file systems to be scheduled for writing out to all file systems.
sync 服务导致内存中的更新文件系统的所有信息被安排写出到所有文件系统。

NOTES: 注意事项：

The writing of data to the file systems is only guaranteed to be scheduled upon return. It is not necessarily complete upon return from sync.
写入数据到文件系统只保证返回后安排。它不一定从 sync 返回之后完成。

5.4.12 mount - Mount a file system

5.4.12 mount - 挂载文件系统

CALLING SEQUENCE: 调用序列：

```
#include <libio.h>

int mount(
   rtems_filesystem_mount_table_entry_t **mt_entry,
   rtems_filesystem_operations_table    *fs_ops,
   rtems_filesystem_options_t        fsoptions,
   char    *device,
   char    *mount_point
);
```

STATUS CODES: 状态码：

EXXX

DESCRIPTION: 描述：

The mount routines mounts the filesystem class which uses the filesystem operations specified by fs_ops and fsoptions. The filesystem is mounted at the directory mount_point and the mode of the mounted filesystem is specified by fsoptions. If this filesystem class requires a device, then the name of the device must be specified by device.
mount 例程挂载文件系统类，使用由 fs_ops 和 fsoptions 指定的文件系统操作。文件系统被挂载在目录 mount_point 上，已挂载文件系统的模式由 fsoptions 修改。如果此文件系统类需要一个设备，则设备名称必须由 device 指定。

If this operation succeeds, the mount table entry for the mounted filesystem is returned in mt_entry.
如果此操作成功，已挂载文件系统的挂载表项在 mt_entry 中返回。

NOTES: 注意事项：

NONE
无

5.4.13 unmount - Unmount file systems

5.4.13 unmount - 卸载文件系统

CALLING SEQUENCE: 调用序列：

```
#include <libio.h>

int unmount(
   const char *mount_path
);
```

STATUS CODES: 状态码：

EXXX

DESCRIPTION: 描述：

The unmount routine removes the attachment of the filesystem specified by mount_path.
unmount 例程移除由 mount_path 指定的文件系统连接。

NOTES: 注意事项：

NONE
无

5.4.14 readv - Vectored read from a file

5.4.14 readv - 从文件定向读取

CALLING SEQUENCE: 调用序列：

```
#include <sys/uio.h>

ssize_t readv(
   int   fd,
   const struct iovec *iov,
   int   iovcnt
);
```

STATUS CODES: 状态码：

In addition to the errors detected by Input and Output Primitives Manager read - Reads from a file, read(), this routine may return -1 and sets errno based upon the following errors:
除了通过“输入和输出原始管理器”的读取——从文件读取的 read()检测错误，此例程可能返回-1 并根据以下错误设置 errno：

EINVAL The sum of the iov_len values in the iov array overflowed an ssize_t.
EINVAL iov 数组中的 iov_len 值的总和溢出了一个 ssize_t 大小。

EINVAL The iovcnt argument was less than or equal to 0, or greater than IOV_MAX.
EINVAL iovcnt 参数小于或等于 0，或大于 IOV_MAX。

DESCRIPTION: 描述：

The readv() function is equivalent to read() except as described here. The readv() function shall place the input data into the iovcnt buffers specified by the members of the iov array: iov[0], iov[1], ..., iov[iovcnt-1].
readv()函数相当于 read()，除了此处所述。readv()函数将放置输入数据到 iov 数组成员：iov[0], iov[1], ..., iov[iovcnt-1]所指定的 iovcnt 缓冲区。

Each iovec entry specifies the base address and length of an area in memory where data should be placed. The readv() function always fills an area completely before proceeding to the next.
每个 iovec 项指定放置数据的内存中的一个区域的基址和长度。readv()函数总是

在继续下一个区域之前完整地填充一个区域。

NOTES: 注意事项：

NONE
无

5.4.15 writev - Vectored write to a file

5.4.15 writev - 定向写入到文件

CALLING SEQUENCE: 调用序列：

```
#include <sys/uio.h>

ssize_t writev(
   int    fd,
   const struct iovec *iov,
   int    iovcnt
);
```

STATUS CODES: 状态码：

In addition to the errors detected by Input and Output Primitives Manager write - Write to a file, write(), this routine may return -1 and sets errno based upon the following errors:
除了通过“输入和输出原始管理器”的写入——写入到文件的 write()检测错误，此例程可能返回-1 并根据以下错误设置 errno：

EINVAL	The sum of the iov_len values in the iov array overflowed an ssize_t.
EINVAL	iov 数组中的 iov_len 值的总和溢出了一个 ssize_t 大小。
EINVAL	The iovcnt argument was less than or equal to 0, or greater than IOV_MAX.
EINVAL	iovcnt 参数小于或等于 0，或大于 IOV_MAX。

DESCRIPTION: 描述：

The writev() function is equivalent to write(), except as noted here. The writev() function gathers output data from the iovcnt buffers specified by the members of the iov array: iov[0], iov[1], ..., iov[iovcnt-1]. The iovcnt argument is valid if greater than 0 and less than or equal to IOV_MAX.
writev()函数相当于 write()，除了此处注释。writev()函数从 iov 数组：iov[0], iov[1], ..., iov[iovcnt-1]所指定的 iovcnt 缓冲区收集输出数据。如果 iovcnt 参数大于 0 并小于或等于 IOV_MAX，则是有效的。

Each iovec entry specifies the base address and length of an area in memory from which data should be written. The writev() function always writes a complete area

before proceeding to the next.
每个 iovec 项指定写入数据的内存中的一个区域的基址和长度。writev()函数总是在继续下一个区域之前写入一个完整的区域。

If fd refers to a regular file and all of the iov_len members in the array pointed to by iov are 0, writev() returns 0 and has no other effect. For other file types, the behavior is unspecified by POSIX.
如果 fd 引用了常规文件,并且数组中的所有 iov_len 成员由 iov 指向 0,则 writev() 返回 0，并且没有其他影响。对于其他文件类型，该行为未被 POSIX 指定。

NOTES: 注意事项：

NONE
无

5.4.16 aio_read - Asynchronous Read

5.4.16 aio_read - 异步读取

CALLING SEQUENCE: 调用序列：

```
int aio_read(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.17 aio_write - Asynchronous Write

5.4.17 aio_write - 异步写入

CALLING SEQUENCE: 调用序列：

```
int aio_write(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.18 lio_listio - List Directed I/O

5.4.18 lio_listio - I/O 所指向的列表

CALLING SEQUENCE: **调用序列：**

```
int lio_listio(
);
```

STATUS CODES: **状态码：**

E The

DESCRIPTION: **描述：**

NOTES: **注意事项：**

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.19 aio_error - Retrieve Error Status of Asynchronous I/O Operation

5.4.19 aio_error - 检索异步 I/O 操作的错误状态

CALLING SEQUENCE: 调用序列：

```
int aio_error(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.20 aio_return - Retrieve Return Status Asynchronous I/O Operation

5.4.20 aio_return - 检索异步 I/O 操作的返回状态

CALLING SEQUENCE: 调用序列：

```
int aio_return(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.21 aio_cancel - Cancel Asynchronous I/O Request

5.4.21 aio_cancel - 取消异步 I/O 请求

CALLING SEQUENCE: 调用序列：

```
int aio_cancel(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.22 aio_suspend - Wait for Asynchronous I/O Request

5.4.22 aio_suspend - 等待异步 I/O 请求

CALLING SEQUENCE: 调用序列：

```
int aio_suspend(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

5.4.23 aio_fsync - Asynchronous File Synchronization

5.4.23 aio_fsync - 异步文件同步化

CALLING SEQUENCE: 调用序列：

```
int aio_fsync(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

6 Device- and Class- Specific Functions Manager

6 设备特定的和类特定的功能管理器

6.1 Introduction

6.1 简介

The device- and class- specific functions manager is ...
设备特定的和类特定的功能管理器是……

The directives provided by the device- and class- specific functions manager are:
由设备特定的和类特定的功能管理器提供的指令有：

- cfgetispeed - Reads terminal input baud rate
- cfgetospeed - Reads terminal output baud rate
- cfsetispeed - Sets terminal input baud rate
- cfsetospeed - Set terminal output baud rate
- tcgetattr - Gets terminal attributes
- tcsetattr - Set terminal attributes
- tcsendbreak - Sends a break to a terminal
- tcdrain - Waits for all output to be transmitted to the terminal
- tcflush - Discards terminal data
- tcflow - Suspends/restarts terminal output
- tcgetpgrp - Gets foreground process group ID
- tcsetpgrp - Sets foreground process group ID

- cfgetispeed - 读取终端输入波特率
- cfgetospeed - 读取终端输出波特率
- cfsetispeed - 设置终端输入波特率
- cfsetospeed - 设置终端输出波特率
- tcgetattr - 获得终端属性
- tcsetattr - 设置终端属性
- tcsendbreak - 发送暂停到终端
- tcdrain - 等待所有输出被传输到终端
- tcflush - 丢弃终端数据
- tcflow - 挂起/重启终端输出
- tcgetpgrp - 获得前台进程组 ID
- tcsetpgrp - 设置前台进程组 ID

6.2 Background

6.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

6.3 Operations

6.3 操作

There is currently no text in this section.
当前在本节中没有文本。

6.4 Directives

6.4 指令

This section details the device- and class- specific functions manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了设备特定的和类特定的功能管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

6.4.1 cfgetispeed - Reads terminal input baud rate

6.4.1 cfgetispeed - 读取终端输入波特率

CALLING SEQUENCE: 调用序列：

```
#include <termios.h>

int cfgetispeed(
    const struct termios *p
);
```

STATUS CODES: 状态码：

The cfgetispeed() function returns a code for baud rate.
cfgetispeed()函数返回波特率的代码。

DESCRIPTION: 描述：

The cfgetispeed() function stores a code for the terminal speed stored in a struct termios. The codes are defined in <termios.h> by the macros BO, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, and B38400.
cfsetispeed()函数存储 struct termios 中所存储的终端速度代码。该代码是通过宏 BO、B50、B75、B110、B134、B150、B200、B300、B600、B1200、B1800、B2400、B4800、B9600、B19200 和 B38400 在<termios.h>中定义的。

The cfgetispeed() function does not do anything to the hardware. It merely stores a value for use by tcsetattr().
cfsetispeed()函数不对硬件做任何事情。它只是存储由 tcsetattr()使用的值。

NOTES: 注意事项：

Baud rates are defined by symbols, such as B110, B1200, B2400. The actual number returned for any given speed may change from system to system.
波特率由符号定义，如 B110、B1200、B2400。对于任何给定的速度，返回的实际数值可能在不同系统之间变化。

6.4.2 cfgetospeed - Reads terminal output baud rate

6.4.2 cfgetospeed - 读取终端输出波特率

CALLING SEQUENCE: 调用序列：

```
#include <termios.h>

int cfgetospeed(
    const struct termios *p
);
```

STATUS CODES: 状态码：

The cfgetospeed() function returns the termios code for the baud rate.
cfgetospeed()函数返回波特率的 termios 代码。

DESCRIPTION: 描述：

The cfgetospeed() function returns a code for the terminal speed stored in a struct termios. The codes are defined in <termios.h> by the macros BO, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, and B38400.
cfgetospeed()函数返回 struct termios 中所存储的终端速度代码。该代码是通过宏 BO、B50、B75、B110、B134、B150、B200、B300、B600、B1200、B1800、B2400、B4800、B9600、B19200 和 B38400 在<termios.h>中定义的。

The cfgetospeed() function does not do anything to the hardware. It merely returns the value stored by a previous call to tcgetattr().
cfgetospeed()函数不对硬件做任何事情。它只是返回由先前调用 tcsetattr()存储的值。

NOTES: 注意事项：

Baud rates are defined by symbols, such as B110, B1200, B2400. The actual number returned for any given speed may change from system to system.
波特率由符号定义，如 B110、B1200、B2400。对于任何给定的速度，返回的实际数值可能在不同系统之间变化。

6.4.3 cfsetispeed - Sets terminal input baud rate

6.4.3 cfsetispeed - 设置终端输入波特率

CALLING SEQUENCE: 调用序列：

```
#include <termios.h>

int cfsetispeed(
   struct termios *p,
   speed_t    speed
);
```

STATUS CODES: 状态码：

The cfsetispeed() function returns a zero when successful and returns -1 when an error occurs.
当成功时，cfsetispeed()函数返回 0，当出现错误时，返回-1。

DESCRIPTION: 描述：

The cfsetispeed() function stores a code for the terminal speed stored in a struct termios. The codes are defined in <termios.h> by the macros B0, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, and B38400.
cfsetispeed()函数存储 struct termios 中所存储的终端速度代码。该代码是通过宏 BO、B50、B75、B110、B134、B150、B200、B300、B600、B1200、B1800、B2400、B4800、B9600、B19200 和 B38400 在<termios.h>中定义的。

NOTES: 注意事项：

This function merely stores a value in the termios structure. It does not change the terminal speed until a tcsetattr() is done. It does not detect impossible terminal speeds.
这个函数只是在 termios 结构中存储一个值。直到完成 tcsetattr()，它才改变终端速度。它不检测不可能的终端速度。

6.4.4 cfsetospeed - Sets terminal output baud rate

6.4.4 cfsetospeed - 设置终端输出波特率

CALLING SEQUENCE: 调用序列：

```
#include <termios.h>

int cfsetospeed(
   struct termios *p,
   speed_t    speed
);
```

STATUS CODES: 状态码：

The cfsetospeed() function returns a zero when successful and returns -1 when an error occurs.
当成功时，cfsetospeed()函数返回 0，当出现错误时，返回-1。

DESCRIPTION: 描述：

The cfsetospeed() function stores a code for the terminal speed stored in a struct termios. The codes are defined in <termios.h> by the macros B0, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, and B38400.
cfsetospeed()函数存储 struct termios 中所存储的终端速度代码。该代码是通过宏 BO、B50、B75、B110、B134、B150、B200、B300、B600、B1200、B1800、B2400、B4800、B9600、B19200 和 B38400 在<termios.h>中定义的。

The cfsetospeed() function does not do anything to the hardware. It merely stores a value for use by tcsetattr().
cfsetospeed()函数不对硬件做任何事情。它只是存储由 tcsetattr()使用的值。

NOTES: 注意事项：

This function merely stores a value in the termios structure. It does not change the terminal speed until a tcsetattr() is done. It does not detect impossible terminal speeds.
这个函数只是在 termios 结构中存储一个值。直到完成 tcsetattr()，它才改变终端速度。它不检测不可能的终端速度。

6.4.5 tcgetattr - Gets terminal attributes

6.4.5 tcgetattr - 获得终端属性

CALLING SEQUENCE: 调用序列：

```
#include <termios.h>
#include <unistd.h>

int tcgetattr(
   int    fildes,
   struct termios *p
);
```

STATUS CODES: 状态码：

EBADF Invalid file descriptor
EBADF 无效的文件描述符

ENOOTY Terminal control function attempted for a file that is not a terminal.
ENOOTY 终端控制函数试图对不是终端的文件进行操作。

DESCRIPTION: 描述：

The tcgetattr() gets the parameters associated with the terminal referred to by fildes and stores them into the termios~~()~~ structure pointed to by termios_p.
tcgetattr()获得与 fildes 所引用的终端相关联的参数，并将它们存储在 termios_p 所指向的 termios 结构。

NOTES: 注意事项：

NONE
无

6.4.6 tcsetattr - Set terminal attributes

6.4.6 tcsetattr - 设置终端属性

CALLING SEQUENCE: 调用序列：

```
#include <termios.h>
#include <unistd.h>

int tcsetattr(
   int    fildes,
   int    options,
   const struct termios *tp
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

6.4.7 tcsendbreak - Sends a break to a terminal

6.4.7 tcsendbreak - 发送暂停到终端

CALLING SEQUENCE: 调用序列：

```
int tcsendbreak(
   int fd
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

6.4.8 tcdrain - Waits for all output to be transmitted to the terminal.

6.4.8 tcdrain - 等待所有输出被传输到终端

CALLING SEQUENCE: 调用序列：

```
#include <termios.h>
#include <unistd.h>

int tcdrain(
    int fildes
);
```

STATUS CODES: 状态码：

EBADF Invalid file descriptor
EBADF 无效的文件描述符

EINTR Function was interrupted by a signal
EINTR 函数被一个信号中断。

ENOTTY Terminal control function attempted for a file that is not a terminal.
ENOOTY 终端控制函数试图对不是终端的文件进行操作。

DESCRIPTION: 描述：

The tcdrain() function waits until all output written to fildes has been transmitted.
tcdrain()函数将等待，直到写入到 fildes 的所有输出已经被传输。

NOTES: 注意事项：

NONE
无

6.4.9 tcflush - Discards terminal data

6.4.9 tcflush - 丢弃终端数据

CALLING SEQUENCE: 调用序列：

```
int tcflush(
   int fd
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

6.4.10 tcflow - Suspends/restarts terminal output.

6.4.10 tcflow - 挂起/重启终端输出

CALLING SEQUENCE: 调用序列：

```
int tcflow(
   int fd
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

6.4.11 tcgetpgrp - Gets foreground process group ID

6.4.11 tcgetpgrp - 获得前台进程组 ID

CALLING SEQUENCE: 调用序列：

```
int tcgetpgrp(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

6.4.12 tcsetpgrp - Sets foreground process group ID

6.4.12 tcsetpgrp - 设置前台进程组 ID

CALLING SEQUENCE: 调用序列：

```
int tcsetpgrp(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine is not currently supported by RTEMS but could be in a future version.
这个例程当前不被 RTEMS 支持，但可能存在于未来的版本中。

7 Language-Specific Services for the C Programming Language Manager

7 适用于 C 编程语言管理器的语言特定的服务

7.1 Introduction

7.1 简介

The language-specific services for the C programming language manager is ...
适用于 C 编程语言管理器的语言特定的服务是……

The directives provided by the language-specific services for the C programming language manager are:
由适用于 C 编程语言管理器的语言特定的服务提供的指令有：

- setlocale - Set the Current Locale
- fileno - Obtain File Descriptor Number for this File
- fdopen - Associate Stream with File Descriptor
- flockfile - Acquire Ownership of File Stream
- ftrylockfile - Poll to Acquire Ownership of File Stream
- funlockfile - Release Ownership of File Stream
- getc_unlocked - Get Character without Locking
- getchar_unlocked - Get Character from stdin without Locking
- putc_unlocked - Put Character without Locking
- putchar_unlocked - Put Character to stdin without Locking
- setjmp - Save Context for Non-Local Goto
- longjmp - Non-Local Jump to a Saved Context
- sigsetjmp - Save Context with Signal Status for Non-Local Goto
- siglongjmp - Non-Local Jump with Signal Status to a Saved Context
- tzset - Initialize Time Conversion Information
- strtok_r - Reentrant Extract Token from String
- asctime_r - Reentrant struct tm to ASCII Time Conversion
- ctime_r - Reentrant time_t to ASCII Time Conversion
- gmtime_r - Reentrant UTC Time Conversion
- localtime_r - Reentrant Local Time Conversion
- rand_r - Reentrant Random Number Generation

- setlocale - 设置当前的位置
- fileno - 获取此文件的文件描述符编号
- fdopen - 将流与文件描述符关联
- flockfile - 获得文件流的所有权
- ftrylockfile - 轮询获得文件流的所有权
- funlockfile - 释放文件流的所有权
- getc_unlocked - 在未锁定的情况下获得字符
- getchar_unlocked - 在未锁定的情况下从 stdin 获得字符
- putc_unlocked - 在未锁定的情况下放置字符
- putchar_unlocked - 在未锁定的情况下放置字符到 stdin
- setjmp - 保存非本地跳转的上下文
- longjmp - 非本地跳转到保存的上下文
- sigsetjmp - 保存非本地跳转的具有信号状态的上下文
- siglongjmp - 使用信号状态非本地跳转到保存的上下文
- tzset - 初始化时间转换信息
- strtok_r - 可重入地提取来自字符串的令牌
- asctime_r - 可重入的 struct tm 到 ASCII 时间的转换
- ctime_r - 可重入的 time_t 到 ASCII 时间的转换
- gmtime_r - 可重入的 UTC 时间转换
- localtime_r - 可重入的本地时间转换
- rand_r - 可重入的随机数发生器

7.2 Background

7.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

7.3 Operations

7.3 操作

There is currently no text in this section.
当前在本节中没有文本。

7.4 Directives

7.4 指令

This section details the language-specific services for the C programming language manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了适用于 C 编程语言管理器的语言特定的服务的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

7.4.1 setlocale - Set the Current Locale

7.4.1 setlocale - 设置当前的位置

CALLING SEQUENCE: 调用序列：

```
int setlocale(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.2 fileno - Obtain File Descriptor Number for this File

7.4.2 fileno - 获取此文件的文件描述符编号

CALLING SEQUENCE: 调用序列：

```
int fileno(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.3 fdopen - Associate Stream with File Descriptor

7.4.3 fdopen - 将流与文件描述符关联

CALLING SEQUENCE: 调用序列：

```
int fdopen(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.4 flockfile - Acquire Ownership of File Stream

7.4.4 flockfile - 获得文件流的所有权

CALLING SEQUENCE: 调用序列：

```
int flockfile(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.5 ftrylockfile - Poll to Acquire Ownership of File Stream

7.4.5 ftrylockfile - 轮询获得文件流的所有权

CALLING SEQUENCE: 调用序列：

```
int ftrylockfile(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.6 funlockfile - Release Ownership of File Stream

7.4.6 funlockfile - 释放文件流的所有权

CALLING SEQUENCE: 调用序列：

```
int funlockfile(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.7 getc_unlocked - Get Character without Locking

7.4.7 getc_unlocked - 在未锁定的情况下获得字符

CALLING SEQUENCE: 调用序列：

```
int getc_unlocked(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.8 getchar_unlocked - Get Character from stdin without Locking

7.4.8 getchar_unlocked - 在未锁定的情况下从 stdin 获得字符

CALLING SEQUENCE: 调用序列：

```
int getchar_unlocked(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.9 putc_unlocked - Put Character without Locking

7.4.9 putc_unlocked - 在未锁定的情况下放置字符

CALLING SEQUENCE: 调用序列：

```
int putc_unlocked(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.10 putchar_unlocked - Put Character to stdin without Locking

7.4.10 putchar_unlocked - 在未锁定的情况下放置字符到 stdin

CALLING SEQUENCE: 调用序列：

```
int putchar_unlocked(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.11 setjmp - Save Context for Non-Local Goto

7.4.11 setjmp - 保存非本地跳转的上下文

CALLING SEQUENCE: 调用序列：

```
int setjmp(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.12 longjmp - Non-Local Jump to a Saved Context

7.4.12 longjmp - 非本地跳转到保存的上下文

CALLING SEQUENCE: 调用序列：

```
int longjmp(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.13 sigsetjmp - Save Context with Signal Status for Non-Local Goto

7.4.13 sigsetjmp - 保存非本地跳转的具有信号状态的上下文

CALLING SEQUENCE: 调用序列：

```
int sigsetjmp(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.14 siglongjmp - Non-Local Jump with Signal Status to a Saved Context

7.4.14 siglongjmp - 使用信号状态非本地跳转到保存的上下文

CALLING SEQUENCE: 调用序列：

```
int siglongjmp(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.15 tzset - Initialize Time Conversion Information

7.4.15 tzset - 初始化时间转换信息

CALLING SEQUENCE: 调用序列：

```
int tzset(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.16 strtok_r - Reentrant Extract Token from String

7.4.16 strtok_r - 可重入地提取来自字符串的令牌

CALLING SEQUENCE: 调用序列：

```
int strtok_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.17 asctime_r - Reentrant struct tm to ASCII Time Conversion

7.4.17 asctime_r - 可重入的 struct tm 到 ASCII 时间的转换

CALLING SEQUENCE: 调用序列：

```
int asctime_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.18 ctime_r - Reentrant time_t to ASCII Time Conversion

7.4.18 ctime_r - 可重入的 time_t 到 ASCII 时间的转换

CALLING SEQUENCE: 调用序列：

```
int ctime_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.19 gmtime_r - Reentrant UTC Time Conversion

7.4.19 gmtime_r - 可重入的 UTC 时间转换

CALLING SEQUENCE: 调用序列：

```
int gmtime_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.20 localtime_r - Reentrant Local Time Conversion

7.4.20 localtime_r - 可重入的本地时间转换

CALLING SEQUENCE: 调用序列：

```
int localtime_r(
);
```

STATUS CODES: 状态码：

E	The

DESCRIPTION: 描述：

NOTES: 注意事项：

7.4.21 rand_r - Reentrant Random Number Generation

7.4.21 rand_r - 可重入的随机数发生器

CALLING SEQUENCE: 调用序列：

```
int rand_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8 System Databases Manager

8 系统数据库管理器

8.1 Introduction

8.1 简介

The system databases manager is ...
系统数据库管理器是……

The directives provided by the system databases manager are:
由系统数据库管理器提供的指令有：

- getgrgid - Get Group File Entry for ID
- getgrgid_r - Reentrant Get Group File Entry for ID
- getgrnam - Get Group File Entry for Name
- getgrnam_r - Reentrant Get Group File Entry for Name
- getpwuid - Get Password File Entry for UID
- getpwuid_r - Reentrant Get Password File Entry for UID
- getpwnam - Get Password File Entry for Name
- getpwnam_r - Reentrant Get Password File Entry for Name

- getgrgid - 获得组文件的 ID 项
- getgrgid_r - 可重入地获得组文件的 ID 项
- getgrnam - 获得组文件的名称项
- getgrnam_r - 可重入地获得组文件的名称项
- getpwuid - 获得密码文件的 UID 项
- getpwuid_r - 可重入地获得密码文件的 UID 项
- getpwnam - 获得密码文件的名称项
- getpwnam_r - 可重入地获得密码文件的名称项

8.2 Background

8.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

8.3 Operations

8.3 操作

There is currently no text in this section.
当前在本节中没有文本。

8.4 Directives

8.4 指令

This section details the system databases manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了系统数据库管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

8.4.1 getgrgid - Get Group File Entry for ID

8.4.1 getgrgid - 获得组文件的 ID 项

CALLING SEQUENCE: 调用序列：

```
int getgrgid(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8.4.2 getgrgid_r - Reentrant Get Group File Entry for ID

8.4.2 getgrgid_r - 可重入地获得组文件的 ID 项

CALLING SEQUENCE: 调用序列：

```
int getgrgid_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8.4.3 getgrnam - Get Group File Entry for Name

8.4.3 getgrnam - 获得组文件的名称项

CALLING SEQUENCE: 调用序列：

```
int getgrnam(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8.4.4 getgrnam_r - Reentrant Get Group File Entry for Name

8.4.4 getgrnam_r - 可重入地获得组文件的名称项

CALLING SEQUENCE: 调用序列：

```
int getgrnam_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8.4.5 getpwuid - Get Password File Entry for UID

8.4.5 getpwuid - 获得密码文件的 UID 项

CALLING SEQUENCE: 调用序列：

```
int getpwuid(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8.4.6 getpwuid_r - Reentrant Get Password File Entry for UID

8.4.6 getpwuid_r - 可重入地获得密码文件的 UID 项

CALLING SEQUENCE: 调用序列：

```
int getpwuid_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8.4.7 getpwnam - Password File Entry for Name

8.4.7 getpwnam - 获得密码文件的名称项

CALLING SEQUENCE: 调用序列：

```
int getpwnam(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

8.4.8 getpwnam_r - Reentrant Get Password File Entry for Name

8.4.8 getpwnam_r - 可重入地获得密码文件的名称项

CALLING SEQUENCE: 调用序列：

```
int getpwnam_r(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

9 Semaphore Manager

9 信号量管理器

9.1 Introduction

9.1 简介

The semaphore manager provides functions to allocate, delete, and control semaphores. This manager is based on the POSIX 1003.1 standard.
信号量管理器提供函数以分配、删除和控制信号量。此管理器基于 POSIX 1003.1 标准。

The directives provided by the semaphore manager are:
由信号量管理器提供的指令有：

- sem_init - Initialize an unnamed semaphore
- sem_destroy - Destroy an unnamed semaphore
- sem_open - Open a named semaphore
- sem_close - Close a named semaphore
- sem_unlink - Remove a named semaphore
- sem_wait - Lock a semaphore
- sem_trywait - Lock a semaphore
- sem_timedwait - Wait on a Semaphore for a Specified Time
- sem_post - Unlock a semaphore
- sem_getvalue - Get the value of a semaphore

- sem_init - 初始化未指定名称的信号量
- sem_destroy - 销毁未指定名称的信号量
- sem_open - 打开指定名称的信号量
- sem_close - 关闭指定名称的信号量
- sem_unlink - 移除指定名称的信号量
- sem_wait - 锁定信号量
- sem_trywait - 锁定信号量
- sem_timedwait - 等待信号量指定的时间
- sem_post - 解锁信号量
- sem_getvalue - 获得信号量的值

9.2 Background

9.2 背景知识

9.2.1 Theory

9.2.1 理论

Semaphores are used for synchronization and mutual exclusion by indicating the availability and number of resources. The task (the task which is returning resources) notifying other tasks of an event increases the number of resources held by the semaphore by one. The task (the task which will obtain resources) waiting for the event decreases the number of resources held by the semaphore by one. If the number of resources held by a semaphore is insufficient (namely 0), the task requiring resources will wait until the next time resources are returned to the semaphore. If there is more than one task waiting for a semaphore, the tasks will be placed in the queue.
信号量用于同步和互斥，表示资源的可用性和数量。任务（正在返回资源的任务）通知事件的其他任务将信号量持有的资源数量增加 1。任务（将要获取资源的任务）等待事件将信号量持有的资源数量减少 1。如果信号量持有的资源数量不足（即 0），需要资源的任务将等待，直到下次资源被返回到信号量。如果有不止一个的任务等待信号量，这些任务将被放置在队列中。

9.2.2 "sem_t" Structure

9.2.2 "sem_t"结构

The sem_t structure is used to represent semaphores. It is passed as an argument to the semaphore directives and is defined as follows:
sem_t 结构用于代表信号量。它被作为一个参数传递给信号量指令，定义如下：

```
typedef int sem_t;
```

9.2.3 Building a Semaphore Attribute Set

9.2.3 建立信号量属性集

9.3 Operations

9.3 操作

9.3.1 Using as a Binary Semaphore

9.3.1 用作二进制信号量

Although POSIX supports mutexes, they are only visible between threads. To work between processes, a binary semaphore must be used.
虽然 POSIX 支持互斥体，它们只在线程之间可见。要在进程之间工作，必须使用二进制信号量。

Creating a semaphore with a limit on the count of 1 effectively restricts the semaphore to being a binary semaphore. When the binary semaphore is available, the count is 1. When the binary semaphore is unavailable, the count is 0.
创建具有计数到 1 的限制的信号量，将该信号量有效地限定为一个二进制信号量。当二进制信号量可用时，计数为 1。当二进制信号量不可用时，计数为 0。

Since this does not result in a true binary semaphore, advanced binary features like the Priority Inheritance and Priority Ceiling Protocols are not available.
因为这不会导致真正的二进制信号量，高级二进制特征，如优先级继承和优先级置顶协议，不可用。

~~There is currently no text in this section.~~

9.4 Directives

9.4 指令

This section details the semaphore manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了信号量管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

9.4.1 sem_init - Initialize an unnamed semaphore

9.4.1 sem_init - 初始化未指定名称的信号量

CALLING SEQUENCE: 调用序列：

```
int sem_init(
   sem_t    *sem,
   int    pshared,
   unsigned int    value
);
```

STATUS CODES: 状态码：

EINVAL The value argument exceeds SEM_VALUE_MAX
EINVAL value 参数超过了 SEM_VALUE_MAX。

ENOSPC A resource required to initialize the semaphore has been exhausted The limit on semaphores (SEM_VALUE_MAX) has been reached
ENOSPC 初始化信号所需的资源已耗尽，已达到对信号量的限制（SEM_VALUE_MAX）。

ENOSYS The function sem_init is not supported by this implementation
ENOSYS 函数 sem_init 不被此实现支持。

EPERM The process lacks appropriate privileges to initialize the semaphore
EPERM 进程缺乏适当的特权来初始化信号量。

DESCRIPTION: 描述：

The sem_init function is used to initialize the unnamed semaphore referred to by "sem". The value of the initialized semaphore is the parameter "value". The semaphore remains valid until it is destroyed.
sem_init 函数用于初始化由 sem 引用的未指定名称的信号量。已初始化信号量的值是参数 value。信号量保持有效，直到它被销毁。

ADD MORE HERE XXX
在这里添加更多 XXX

NOTES: 注意事项：

If the functions completes successfully, it shall return a value of zero. Otherwise, it shall return a value of -1 and set "errno" to specify the error that occurred.
如果函数成功完成，它应返回 0 值。否则，它应返回-1 值并设置 errno，以指定出现的错误。

Multiprocessing is currently not supported in this implementation.
当前在此实现中不支持多处理。

9.4.2 sem_destroy - Destroy an unnamed semaphore

9.4.2 sem_destroy - 销毁未指定名称的信号量

CALLING SEQUENCE: 调用序列：

```
int sem_destroy(
   sem_t *sem
);
```

STATUS CODES: 状态码：

EINVAL The value argument exceeds SEM_VALUE_MAX
EINVAL value 参数超过了 SEM_VALUE_MAX。

ENOSYS The function sem_init is not supported by this implementation
ENOSYS 函数 sem_init 不被此实现支持。

EBUSY There are currently processes blocked on the semaphore
EBUSY 当前有进程阻塞在信号量上。

DESCRIPTION: 描述：

The sem_destroy function is used to destroy an unnamed semaphore refered to by "sem". sem_destroy can only be used on a semaphore that was created using sem_init.
sem_destroy 函数用于销毁由 sem 引用的未指定名称的信号量。sem_destroy 只能用在使用 sem_init 创建的信号量上。

NOTES: 注意事项：

If the functions completes successfully, it shall return a value of zero. Otherwise, it shall return a value of -1 and set "errno" to specify the error that occurred.
如果函数成功完成，它应返回 0 值。否则，它应返回-1 值并设置 errno，以指定出现的错误。

Multiprocessing is currently not supported in this implementation.
当前在此实现中不支持多处理。

9.4.3 sem_open - Open a named semaphore

9.4.3 sem_open - 打开指定名称的信号量

CALLING SEQUENCE: 调用序列：

```
int sem_open(
   const char *name,
   int    oflag
);
```

ARGUMENTS: 参数：

The following flag bit may be set in oflag:
以下标志位可以在 oflag 中设置：

O_CREAT - Creates the semaphore if it does not already exist. If O_CREAT is set and the semaphore already exists then O_CREAT has no effect. Otherwise, sem_open() creates a semaphore. The O_CREAT flag requires the third and fourth argument: mode and value of type mode_t and unsigned int, respectively.
O_CREAT - 如果该信号量尚未存在，创建它。如果设置了 O_CREAT 并且该信号已存在，则 O_CREAT 不起作用。否则，sem_open()创建一个信号量。O_CREAT 标志需要第三和第四参数：分别是模式和类型 mode_t 和 unsigned int 的值。

O_EXCL - If O_EXCL and O_CREAT are set, all call to sem_open() shall fail if the semaphore name exists
O_EXCL - 如果设置了 O_EXCL 和 O_CREAT，并且该信号量的名称存在，则对 sem_open()的所有调用都应失败。

STATUS CODES: 状态码：

EACCES Valid name specified but oflag permissions are denied, or the semaphore name specified does not exist and permission to create the named semaphore is denied.
EACCES 指定了有效名称，但 oflag 权限被拒绝，或指定的信号量名称不存在并且创建指定名称的信号量的权限被拒绝。

EEXIST O_CREAT and O_EXCL are set and the named semaphore already exists.
EEXIST 设置了 O_CREAT 和 O_EXCL 并且指定名称的信号量已存在。

EINTR The sem_open() operation was interrupted by a signal.
EINTR sem_open()操作被一个信号中断。

EINVAL The sem_open() operation is not supported for the given name.
EINVAL sem_open()操作不支持给定的名称。

EMFILE Too many semaphore descriptors or file descriptors in use by this process.
EMFILE 这个进程使用了太多的信号量描述符或文件描述符。

ENAMETOOLONG The length of the name exceed PATH_MAX or name component is longer than NAME_MAX while POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 名称长度超过 PATH_MAX，或名称部分大于 NAME_MAX 而 POSIX_NO_TRUNC 生效。

ENOENT O_CREAT is not set and the named semaphore does not exist.
ENOENT 未设置 O_CREAT，并且指定名称的信号量不存在。

ENOSPC There is insufficient space for the creation of a new named semaphore.
ENOSPC 没有足够的空间创建新的指定名称的信号量。

ENOSYS The function sem_open() is not supported by this implementation.
ENOSYS 函数 sem_open()不被此实现支持。

DESCRIPTION: 描述：

The sem_open() function establishes a connection between a specified semaphore and a process. After a call to sem_open with a specified semaphore name, a process can reference to semaphore by the associated name using the address returned by the call. The oflag arguments listed above control the state of the semaphore by determining if the semaphore is created or accessed by a call to sem_open().
sem_open()函数建立指定信号量和进程之间的连接。在使用指定的信号量名称调用 sem_open 之后，一个进程可以通过关联的名称使用该调用所返回的地址来引用信号量。通过确认该信号量是否是通过调用 sem_open()来创建或访问，上文所列的 oflag 参数控制信号量的状态。

NOTES: 注意事项：

9.4.4 sem_close - Close a named semaphore

9.4.4 sem_close - 关闭指定名称的信号量

CALLING SEQUENCE: 调用序列：

```
int sem_close(
   sem_t *sem_close
);
```

STATUS CODES: 状态码：

EACCES The semaphore argument is not a valid semaphore descriptor.
EACCES semaphore 参数不是有效的信号量描述符。

ENOSYS The function sem_close is not supported by this implementation.
ENOSYS 函数 sem_close 不被此实现支持。

DESCRIPTION: 描述：

The sem_close() function is used to indicate that the calling process is finished using the named semaphore indicated by sem. The function sem_close deallocates any system resources that were previously allocated by a sem_open system call. If sem_close() completes successfully it returns a 1, otherwise a value of -1 is return and errno is set.
sem_close()函数用于表示调用进程使用由 sem 表示的指定名称的信号量来完成。函数 sem_close 释放先前通过 sem_open 系统调用分配的所有系统资源。如果 sem_close()成功完成，它返回 1，否则返回-1 值并设置 errno。

NOTES: 注意事项：

9.4.5 sem_unlink - Unlink a semaphore

9.4.5 sem_unlink - 断开已命名信号量

CALLING SEQUENCE: 调用序列：

```
int sem_unlink(
   const char *name
);
```

STATUS CODES: 状态码：

EACCESS Permission is denied to unlink a semaphore.
EACCESS 断开信号量的权限被拒绝。

ENAMETOOLONG The length of the strong name exceed NAME_MAX while POSIX_NO_TRUNC is in effect.
ENAMETOOLONG 强名称的长度超过 NAME_MAX，而 POSIX_NO_TRUNC 生效。

ENOENT The name of the semaphore does not exist.
ENOENT 信号量的名称不存在。

ENOSPC There is insufficient space for the creation of a new named semaphore.
ENOSPC 没有足够的空间创建新的指定名称的信号量。

ENOSYS The function sem_unlink is not supported by this implementation.
ENOSYS 函数 sem_unlink 不被此实现支持。

DESCRIPTION: 描述：

The sem_unlink() function shall remove the semaphore name by the string name. If a process is currently accessing the name semaphore, the sem_unlink command has no effect. If one or more processes have the semaphore open when the sem_unlink function is called, the destruction of semaphores shall be postponed until all reference to semaphore are destroyed by calls to sem_close, _exit(), or exec. After all references have been destroyed, it returns immediately.
sem_unlink()函数应移除由字符串 name 指定名称的信号量。如果一个进程当前正在访问指定名称的信号量，sem_unlink 命令不起作用。如果当调用 sem_unlink 函数时，一个或多个进程有信号量打开，则信号量的销毁应推迟，直到所有对信号量引用通过调用 sem_close、_exit()或 exec 来销毁。在所有引用已销毁之后，

它立即返回。

If the termination is successful, the function shall return 0. Otherwise, a -1 is returned and the errno is set.
如果终止成功，该函数应返回 0。否则返回-1 并设置 errno。

NOTES: 注意事项：

9.4.6 sem_wait - Wait on a Semaphore

9.4.6 sem_wait - 等待信号量

CALLING SEQUENCE: 调用序列：

```
int sem_wait(
   sem_t *sem
);
```

STATUS CODES: 状态码：

EINVAL The "sem" argument does not refer to a valid semaphore
EINVAL sem 参数没有引用到有效的信号量。

DESCRIPTION: 描述：

This function attempts to lock a semaphore specified by sem. If the semaphore is available, then the semaphore is locked (i.e., the semaphore value is decremented). If the semaphore is unavailable (i.e., the semaphore value is zero), then the function will block until the semaphore becomes available. It will then successfully lock the semaphore. The semaphore remains locked until released by a sem_post() call.
这个函数尝试锁定由 sem 指定名称的信号量。如果信号量可用，则信号量被锁定（即信号量的值被递减）。如果信号量不可用（即信号量的值为 0），则函数将阻塞，直到信号量变得可用。这时它将成功锁定信号量。信号量保持锁定，直到由 sem_post()调用释放。

If the call is unsuccessful, then the function returns -1 and sets errno to the appropriate error code.
如果调用不成功，则该函数返回-1，并设置 errno 为适当的错误代码。

NOTES: 注意事项：

Multiprocessing is not supported in this implementation.
在此实现中不支持多处理。

9.4.7 sem_trywait - Non-blocking Wait on a Semaphore

9.4.7 sem_trywait - 非阻塞等待信号量

CALLING SEQUENCE: 调用序列：

```
int sem_trywait(
   sem_t *sem
);
```

STATUS CODES: 状态码：

EAGAIN The semaphore is not available (i.e., the semaphore value is zero), so the semaphore could not be locked.
EAGAIN 信号量不可用（即信号量的值为 0），所以不能锁定信号量。

EINVAL The sem argument does not refer to a valid semaphore
EINVAL sem 参数没有引用到有效的信号量。

DESCRIPTION: 描述：

This function attempts to lock a semaphore specified by sem. If the semaphore is available, then the semaphore is locked (i.e., the semaphore value is decremented) and the function returns a value of 0. The semaphore remains locked until released by a sem_post() call. If the semaphore is unavailable (i.e., the semaphore value is zero), then the function will return a value of -1 immediately and set errno to EAGAIN.
这个函数尝试锁定由 sem 指定名称的信号量。如果信号量可用，则信号量被锁定（即信号量的值被递减），函数返回 0 值。信号量保持锁定，直到由 sem_post() 调用释放。如果信号量不可用（即信号量的值为 0），则函数将立即返回-1 值，并设置 errno 为 EAGAIN。

If the call is unsuccessful, then the function returns -1 and sets errno to the appropriate error code.
如果调用不成功，则该函数返回-1，并设置 errno 为适当的错误代码。

NOTES: 注意事项：

Multiprocessing is not supported in this implementation.
在此实现中不支持多处理。

9.4.8 sem_timedwait - Wait on a Semaphore for a Specified Time

9.4.8 sem_timedwait - 等待信号量指定的时间

CALLING SEQUENCE: 调用序列：

```
int sem_timedwait(
   sem_t    *sem,
   const struct timespec *abstime
);
```

STATUS CODES: 状态码：

EAGAIN The semaphore is not available (i.e., the semaphore value is zero), so the semaphore could not be locked.
EAGAIN 信号量不可用（即信号量的值为 0），所以不能锁定信号量。

EINVAL The sem argument does not refer to a valid semaphore
EINVAL sem 参数没有引用到有效的信号量。

DESCRIPTION: 描述：

This function attemtps to lock a semaphore specified by sem, and will wait for the semaphore until the absolute time specified by abstime. If the semaphore is available, then the semaphore is locked (i.e., the semaphore value is decremented) and the function returns a value of 0. The semaphore remains locked until released by a sem_post() call. If the semaphore is unavailable, then the function will wait for the semaphore to become available for the amount of time specified by timeout.
这个函数尝试锁定由 sem 指定名称的信号量，并将于等待信号量，直到由 abstime 指定的绝对时间。如果信号量可用，则信号量被锁定（即信号量的值被递减），函数返回 0 值。信号量保持锁定直到由 sem_post()调用释放。如果信号量不可用，则函数将等待信号量变得可用，等待由 timeout 指定的时间数量。

If the semaphore does not become available within the interval specified by timeout, then the function returns -1 and sets errno to EAGAIN. If any other error occurs, the function returns -1 and sets errno to the appropriate error code.
如果信号量没有在 timeout 所指定的间隔内变得可用，则该函数返回-1 并设置 errno 为 EAGAIN。如果出现任何其他错误，该函数返回-1 并设置 errno 为适当的错误代码。

NOTES:　注意事项：

Multiprocessing is not supported in this implementation.
在此实现中不支持多处理。

9.4.9 sem_post - Unlock a Semaphore

9.4.9 sem_post - 解锁信号量

CALLING SEQUENCE: 调用序列：

```
int sem_post(
   sem_t *sem
);
```

STATUS CODES: 状态码：

EINVAL	The sem argument does not refer to a valid semaphore
EINVAL	sem 参数没有引用到有效的信号量。

DESCRIPTION: 描述：

This function attempts to release the semaphore specified by sem. If other tasks are waiting on the semaphore, then one of those tasks (which one depends on the scheduler being used) is allowed to lock the semaphore and return from its sem_wait(), sem_trywait(), or sem_timedwait() call. If there are no other tasks waiting on the semaphore, then the semaphore value is simply incremented. sem_post() returns 0 upon successful completion.
这个函数尝试释放由 sem 指定名称的信号量。如果其他任务正在等待信号量，则其中一个任务（取决于正在使用的调度器）被允许锁定信号量，并从其 sem_wait()、sem_trywait()或 sem_timedwait()调用返回。如果没有其他任务等待信号量，则信号量的值被简单递增。sem_post()在成功完成之后返回 0。

If an error occurs, the function returns -1 and sets errno to the appropriate error code.
如果出现错误，该函数返回-1 并设置 errno 为适当的错误代码。

NOTES: 注意事项：

Multiprocessing is not supported in this implementation.
在此实现中不支持多处理。

9.4.10 sem_getvalue - Get the value of a semaphore

9.4.10 sem_getvalue - 获得信号量的值

CALLING SEQUENCE: 调用序列：

```
int sem_getvalue(
   sem_t *sem,
   int    *sval
);
```

STATUS CODES: 状态码：

EINVAL The "sem" argument does not refer to a valid semaphore
EINVAL sem 参数没有引用到有效的信号量。

ENOSYS The function sem_getvalue is not supported by this implementation
ENOSYS 函数 sem_getvalue 不被此实现支持。

DESCRIPTION: 描述：

The sem_getvalue functions sets the location referenced by the "sval" argument to the value of the semaphore without affecting the state of the semaphore. The updated value represents a semaphore value that occurred at some point during the call, but is not necessarily the actual value of the semaphore when it returns to the calling process.
sem_getvalue 函数设置由 sval 参数引用的位置为不影响信号量状态的信号量的值。当它返回到调用进程时，更新的值代表在调用期间出现在某些点上的信号量的值，但不一定是信号量的实际值。

If "sem" is locked, the value returned by sem_getvalue will be zero or a negative number whose absolute value is the number of processes waiting for the semaphore at some point during the call.
如果 sem 被锁定，由 sem_getvalue 返回的值将为 0 或负数，它的绝对值是在调用期间在某些点上等待信号量的进程的数量。

NOTES: 注意事项：

If the functions completes successfully, it shall return a value of zero. Otherwise, it shall return a value of -1 and set "errno" to specify the error that occurred.

如果该函数成功完成，它应返回 0 值。否则它应返回-1 值并设置 errno，以指定出现的错误。

10 Mutex Manager

10 互斥体管理器

10.1 Introduction

10.1 简介

The mutex manager implements the functionality required of the mutex manager as defined by POSIX 1003.1b-1996. This standard requires that a compliant operating system provide the facilities to ensure that threads can operate with mutual exclusion from one another and defines the API that must be provided.
互斥体管理器实现由 POSIX 1003.1b-1996 定义的互斥体管理器的所需功能。此标准要求兼容操作系统提供机制，确保线程可以从另一个线程互斥操作，并定义必须提供的 API。

The services provided by the mutex manager are:
由互斥体管理器提供的服务有：

- pthread_mutexattr_init - Initialize a Mutex Attribute Set
- pthread_mutexattr_destroy - Destroy a Mutex Attribute Set
- pthread_mutexattr_setprotocol - Set the Blocking Protocol
- pthread_mutexattr_getprotocol - Get the Blocking Protocol
- pthread_mutexattr_setprioceiling - Set the Priority Ceiling
- pthread_mutexattr_getprioceiling - Get the Priority Ceiling
- pthread_mutexattr_setpshared - Set the Visibility
- pthread_mutexattr_getpshared - Get the Visibility
- pthread_mutex_init - Initialize a Mutex
- pthread_mutex_destroy - Destroy a Mutex
- pthread_mutex_lock - Lock a Mutex
- pthread_mutex_trylock - Poll to Lock a Mutex
- pthread_mutex_timedlock - Lock a Mutex with Timeout
- pthread_mutex_unlock - Unlock a Mutex
- pthread_mutex_setprioceiling - Dynamically Set the Priority Ceiling
- pthread_mutex_getprioceiling - Dynamically Get the Priority Ceiling

- pthread_mutexattr_init - 初始化互斥体属性集
- pthread_mutexattr_destroy - 销毁互斥体属性集
- pthread_mutexattr_setprotocol - 设置阻塞协议

- pthread_mutexattr_getprotocol - 获得阻塞协议
- pthread_mutexattr_setprioceiling - 设置优先级上限
- pthread_mutexattr_getprioceiling - 获得优先级上限
- pthread_mutexattr_setpshared - 设置可见性
- pthread_mutexattr_getpshared - 获得可见性
- pthread_mutex_init - 初始化互斥体
- pthread_mutex_destroy - 销毁互斥体
- pthread_mutex_lock - 锁定互斥体
- pthread_mutex_trylock - 轮询锁定互斥体
- pthread_mutex_timedlock - 锁定具有超时的互斥体
- pthread_mutex_unlock - 解锁互斥体
- pthread_mutex_setprioceiling - 动态设置优先级上限
- pthread_mutex_getprioceiling - 动态获得优先级上限

10.2 Background

10.2 背景知识

10.2.1 Mutex Attributes

10.2.1 互斥体属性

Mutex attributes are utilized only at mutex creation time. A mutex attribute structure may be initialized and passed as an argument to the mutex_init routine. Note that the priority ceiling of a mutex may be set at run-time.
互斥体属性仅在互斥体创建时被利用。互斥体属性结构可以被初始化，并作为参数传递给 mutex_init 例程。请注意，互斥体的优先级上限可以在运行时设置。

blocking protocol is the XXX
阻塞协议 是 XXX

priority ceiling is the XXX
优先级上限 是 XXX

pshared is the XXX
pshared 是 XXX

10.2.2 PTHREAD_MUTEX_INITIALIZER

10.2.2 PTHREAD_MUTEX_INITIALIZER

This is a special value that a variable of type pthread_mutex_t may be statically initialized to as shown below:
这是一个特殊值，是类型 pthread_mutex_t 的变量，可以被静态初始化为如下所示：

```
pthread_mutex_t my_mutex = PTHREAD_MUTEX_INITIALIZER;
```

This indicates that my_mutex will be automatically initialized by an implicit call to pthread_mutex_init the first time the mutex is used.
这表示 my_mutex 将在第一次使用互斥体时通过隐式地调用 pthread_mutex_init 来自动初始化。

Note that the mutex will be initialized with default attributes.
请注意，将使用默认属性初始化互斥体。

10.3 Operations

10.3 操作

There is currently no text in this section.
当前在本节中没有文本。

10.4 Services

10.4 服务

This section details the mutex manager’s services. A subsection is dedicated to each of this manager’s services and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了互斥体管理器的服务。一小节专注于该管理器的一个服务，描述了调用序列、相关的常量、用法和状态码。

10.4.1 pthread_mutexattr_init - Initialize a Mutex Attribute Set

10.4.1 pthread_mutexattr_init - 初始化互斥体属性集

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutexattr_init(
   pthread_mutexattr_t *attr
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。

DESCRIPTION: 描述：

The pthread_mutexattr_init routine initializes the mutex attributes object specified by attr with the default value for all of the individual attributes.
pthread_mutexattr_init 例程使用所有个别属性的默认值初始化由 attr 指定的互斥体属性对象。

NOTES: 注意事项：

XXX insert list of default attributes here.
XXX 在这里插入默认属性的列表。

10.4.2 pthread_mutexattr_destroy - Destroy a Mutex Attribute Set

10.4.2 pthread_mutexattr_destroy - 销毁互斥体属性集

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutexattr_destroy(
   pthread_mutexattr_t *attr
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。

DESCRIPTION: 描述：

The pthread_mutex_attr_destroy routine is used to destroy a mutex attributes object. The behavior of using an attributes object after it is destroyed is implementation dependent.
pthread_mutex_attr_destroy 例程用于销毁互斥体属性对象。在它被销毁之后使用属性对象的行为是实现依赖的。

NOTES: 注意事项：

NONE
无

10.4.3 pthread_mutexattr_setprotocol - Set the Blocking Protocol

10.4.3 pthread_mutexattr_setprotocol - 设置阻塞协议

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutexattr_setprotocol(
   pthread_mutexattr_t *attr,
   int    protocol
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。
EINVAL	The protocol argument is invalid.
EINVAL	协议参数无效。

DESCRIPTION: 描述：

The pthread_mutexattr_setprotocol routine is used to set value of the protocol attribute. This attribute controls the order in which threads waiting on this mutex will receive it.
pthread_mutexattr_setprotocol 例程用于设置 protocol 属性的值。此属性控制等待此互斥体的线程接收它的顺序。

The protocol can be one of the following:
protocol 可以是下列值之一：

PTHREAD_PRIO_NONE

in which case blocking order is FIFO.
在这种情况下阻塞顺序是 FIFO。

PTHREAD_PRIO_INHERIT

in which case blocking order is priority with the priority inheritance protocol in effect.
在这种情况下阻塞顺序实际上是具有优先级继承协议的优先级。

PTHREAD_PRIO_PROTECT

in which case blocking order is priority with the priority ceiling protocol in effect.
在这种情况下阻塞顺序实际上是具有优先级置顶协议的优先级。

NOTES: 注意事项：

There is currently no way to get simple priority blocking ordering with POSIX mutexes even though this could easily by supported by RTEMS.
当前还没有办法使用 POSIX 互斥体获得简单优先级阻塞排序，即使这很容易由 RTEMS 支持。

10.4.4 pthread_mutexattr_getprotocol - Get the Blocking Protocol

10.4.4 pthread_mutexattr_getprotocol - 获得阻塞协议

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutexattr_getprotocol(
   pthread_mutexattr_t *attr,
   int    *protocol
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The protocol pointer argument is invalid.
EINVAL 协议指针参数无效。

DESCRIPTION: 描述：

The pthread_mutexattr_getprotocol routine is used to obtain the value of the protocol attribute. This attribute controls the order in which threads waiting on this mutex will receive it.
pthread_mutexattr_getprotocol 例程用于获取 protocol 属性的值。此属性控制等待此互斥体的线程接收它的顺序。

NOTES: 注意事项：

NONE
无

10.4.5 pthread_mutexattr_setprioceiling - Set the Priority Ceiling

10.4.5 pthread_mutexattr_setprioceiling - 设置优先级上限

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_mutexattr_setprioceiling(
   pthread_mutexattr_t *attr,
   int    prioceiling
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The prioceiling argument is invalid.
EINVAL prioceiling 参数无效。

DESCRIPTION: 描述：

The pthread_mutexattr_setprioceiling routine is used to set value of the prioceiling attribute. This attribute specifies the priority that is the ceiling for threads obtaining this mutex. Any task obtaining this mutex may not be of greater priority that the ceiling. If it is of lower priority, then its priority will be elevated to prioceiling.
pthread_mutexattr_setprioceiling 例程用于设置 prioceiling 属性的值。此属性指定线程获取此互斥体的上限优先级。任何任务获取此互斥体都不能是大于上限的优先级。如果是较低的优先级，则它的优先级将被提高到 prioceiling。

NOTES: 注意事项：

NONE
无

10.4.6 pthread_mutexattr_getprioceiling - Get the Priority Ceiling

10.4.6 pthread_mutexattr_getprioceiling - 获得优先级上限

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_mutexattr_getprioceiling(
   const pthread_mutexattr_t *attr,
   int    *prioceiling
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The prioceiling pointer argument is invalid.
EINVAL prioceiling 指针参数无效。

DESCRIPTION: 描述：

The pthread_mutexattr_getprioceiling routine is used to obtain the value of the prioceiling attribute. This attribute specifies the priority ceiling for this mutex.
pthread_mutexattr_getprioceiling 例程用于获取 prioceiling 属性的值。此属性指定此互斥体的优先级上限。

NOTES: 注意事项：

NONE
无

10.4.7 pthread_mutexattr_setpshared - Set the Visibility

10.4.7 pthread_mutexattr_setpshared - 设置可见性

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_mutexattr_setpshared(
   pthread_mutexattr_t *attr,
   int    pshared
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。
EINVAL	The pshared argument is invalid.
EINVAL	pshared 参数无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.8 pthread_mutexattr_getpshared - Get the Visibility

10.4.8 pthread_mutexattr_getpshared - 获得可见性

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutexattr_getpshared(
   const pthread_mutexattr_t *attr,
   int    *pshared
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The pshared pointer argument is invalid.
EINVAL pshared 参数无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.9 pthread_mutex_init - Initialize a Mutex

10.4.9 pthread_mutex_init - 初始化互斥体

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutex_init(
   pthread_mutex_t    *mutex,
   const pthread_mutexattr_t *attr
);
```

STATUS CODES: 状态码：

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The specified protocol is invalid.
EINVAL 指定的协议无效。

EAGAIN The system lacked the necessary resources to initialize another mutex.
EAGAIN 系统缺少必要的资源以初始化另一个互斥体。

ENOMEM Insufficient memory exists to initialize the mutex.
ENOMEM 现有的内存不足以初始化互斥体。

EBUSY Attempted to reinialize the object reference by mutex, a previously initialized, but not yet destroyed.
EBUSY 试图重新初始化由 mutex 引用的对象，先前已初始化，但尚未销毁。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.10 pthread_mutex_destroy - Destroy a Mutex

10.4.10 pthread_mutex_destroy - 销毁互斥体

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutex_destroy(
   pthread_mutex_t *mutex
);
```

STATUS CODES: 状态码：

EINVAL The specified mutex is invalid.
EINVAL 指定的互斥体无效。

EBUSY Attempted to destroy the object reference by mutex, while it is locked or referenced by another thread.
EBUSY 试图销毁由 mutex 引用的对象，而它被锁定或由另一个线程引用。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.11 pthread_mutex_lock - Lock a Mutex

10.4.11 pthread_mutex_lock - 锁定互斥体

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutex_lock(
   pthread_mutex_t *mutex
);
```

STATUS CODES: 状态码：

EINVAL The specified mutex is invalid.
EINVAL 指定的互斥体无效。

EINVAL The mutex has the protocol attribute of PTHREAD_PRIO_PROTECT and the priority of the calling thread is higher than the current priority ceiling.
EINVAL 互斥体有 PTHREAD_PRIO_PROTECT 的协议属性，调用线程的优先级高于当前的优先级上限。

EDEADLK The current thread already owns the mutex.
EDEADLK 当前线程已拥有互斥体。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.12 pthread_mutex_trylock - Poll to Lock a Mutex

10.4.12 pthread_mutex_trylock - 轮询锁定互斥体

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutex_trylock(
   pthread_mutex_t *mutex
);
```

STATUS CODES: 状态码：

EINVAL The specified mutex is invalid.
EINVAL 指定的互斥体无效。

EINVAL The mutex has the protocol attribute of PTHREAD_PRIO_PROTECT and the priority of the calling thread is higher than the current priority ceiling.
EINVAL 互斥体有 PTHREAD_PRIO_PROTECT 的协议属性，调用线程的优先级高于当前的优先级上限。

EDEADLK The current thread already owns the mutex.
EDEADLK 当前线程已拥有互斥体。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.13 pthread_mutex_timedlock - Lock a Mutex with Timeout

10.4.13 pthread_mutex_timedlock - 锁定具有超时的互斥体

CALLING SEQUENCE: 调用序列：

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

int pthread_mutex_timedlock(
   pthread_mutex_t    *mutex,
   const struct timespec *timeout
);
```

STATUS CODES: 状态码：

EINVAL The specified mutex is invalid.
EINVAL 指定的互斥体无效。

EINVAL The nanoseconds field of timeout is invalid.
EINVAL 超时的纳秒字段无效。

EINVAL The mutex has the protocol attribute of PTHREAD_PRIO_PROTECT and the priority of the calling thread is higher than the current priority ceiling.
EINVAL 互斥体有 PTHREAD_PRIO_PROTECT 的协议属性，调用线程的优先级高于当前的优先级上限。

EDEADLK The current thread already owns the mutex.
EDEADLK 当前线程已拥有互斥体。

ETIMEDOUT The calling thread was unable to obtain the mutex within the specified timeout period.
ETIMEDOUT 调用线程无法在指定的超时周期内获取互斥体。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.14 pthread_mutex_unlock - Unlock a Mutex

10.4.14 pthread_mutex_unlock - 解锁互斥体

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutex_unlock(
   pthread_mutex_t *mutex
);
```

STATUS CODES: 状态码：

EINVAL The specified mutex is invalid.
EINVAL 指定的互斥体无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.15 pthread_mutex_setprioceiling - Dynamically Set the Priority Ceiling

10.4.15 pthread_mutex_setprioceiling - 动态设置优先级上限

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutex_setprioceiling(
   pthread_mutex_t *mutex,
   int    prioceiling,
   int    *oldceiling
);
```

STATUS CODES: 状态码：

EINVAL The oldceiling pointer parameter is invalid.
EINVAL oldceiling 指针参数无效。

EINVAL The prioceiling parameter is an invalid priority.
EINVAL prioceiling 参数是无效的优先级。

EINVAL The specified mutex is invalid.
EINVAL 指定的互斥体无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

10.4.16 pthread_mutex_getprioceiling - Get the Current Priority Ceiling

10.4.16 pthread_mutex_getprioceiling - 动态获得优先级上限

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_mutex_getprioceiling(
   pthread_mutex_t *mutex,
   int    *prioceiling
);
```

STATUS CODES: 状态码：

EINVAL	The prioceiling pointer parameter is invalid.
EINVAL	prioceiling 指针参数无效。
EINVAL	The specified mutex is invalid.
EINVAL	指定的互斥体无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

11 Condition Variable Manager

11 条件变量管理器

11.1 Introduction

11.1 简介

The condition variable manager ...
条件变量管理器……

The directives provided by the condition variable manager are:
由条件变量管理器提供的指令有：

- pthread_condattr_init - Initialize a Condition Variable Attribute Set
- pthread_condattr_destroy - Destroy a Condition Variable Attribute Set
- pthread_condattr_setpshared - Set Process Shared Attribute
- pthread_condattr_getpshared - Get Process Shared Attribute
- pthread_cond_init - Initialize a Condition Variable
- pthread_cond_destroy - Destroy a Condition Variable
- pthread_cond_signal - Signal a Condition Variable
- pthread_cond_broadcast - Broadcast a Condition Variable
- pthread_cond_wait - Wait on a Condition Variable
- pthread_cond_timedwait - Wait with Timeout a Condition Variable

- pthread_condattr_init - 初始化条件变量属性集
- pthread_condattr_destroy - 销毁条件变量属性集
- pthread_condattr_setpshared - 设置进程共享属性
- pthread_condattr_getpshared - 获得进程共享属性
- pthread_cond_init - 初始化条件变量
- pthread_cond_destroy - 销毁条件变量
- pthread_cond_signal - 用信号发送条件变量
- pthread_cond_broadcast - 广播条件变量
- pthread_cond_wait - 等待条件变量
- pthread_cond_timedwait - 超时等待条件变量

11.2 Background

11.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

11.3 Operations

11.3 操作

There is currently no text in this section.
当前在本节中没有文本。

11.4 Directives

11.4 指令

This section details the condition variable manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了条件变量管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

11.4.1 pthread_condattr_init - Initialize a Condition Variable Attribute Set

11.4.1 pthread_condattr_init -初始化条件变量属性集

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_condattr_init(
   pthread_condattr_t    *attr
);
```

STATUS CODES: 状态码：

ENOMEM Insufficient memory is available to initialize the condition variable attributes object.
ENOMEM 没有足够的内存可用于初始化条件变量属性对象。

DESCRIPTION: 描述：

NOTES: 注意事项：

11.4.2 pthread_condattr_destroy - Destroy a Condition Variable Attribute Set

11.4.2 pthread_condattr_destroy - 销毁条件变量属性集

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_condattr_destroy(
   pthread_condattr_t    *attr
);
```

STATUS CODES: 状态码：

EINVAL The attribute object specified is invalid.
EINVAL 指定的属性对象无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

11.4.3 pthread_condattr_setpshared - Set Process Shared Attribute

11.4.3 pthread_condattr_setpshared - 设置进程共享属性

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_condattr_setpshared(
    pthread_condattr_t *attr,
    int    pshared
);
```

STATUS CODES: 状态码：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

NOTES: 注意事项：

11.4.4 pthread_condattr_getpshared - Get Process Shared Attribute

11.4.4 pthread_condattr_getpshared - 获得进程共享属性

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_condattr_getpshared(
   const pthread_condattr_t *attr,
   int    *pshared
);
```

STATUS CODES: 状态码：

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

NOTES: 注意事项：

11.4.5 pthread_cond_init - Initialize a Condition Variable

11.4.5 pthread_cond_init - 初始化条件件变量

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_cond_init(
   pthread_cond_t    *cond,
   const pthread_condattr_t *attr
);
```

STATUS CODES: 状态码：

EAGAIN The system lacked a resource other than memory necessary to create the initialize the condition variable object.
EAGAIN 系统创建初始化条件变量对象，除了必要的内存，还缺乏一个资源。

ENOMEM Insufficient memory is available to initialize the condition variable object.
ENOMEM 没有足够的内存可用于初始化条件变量对象。

EBUSY The specified condition variable has already been initialized.
EBUSY 指定的条件变量已经被初始化。

EINVAL The specified attribute value is invalid.
EINVAL 指定的属性值无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

11.4.6 pthread_cond_destroy - Destroy a Condition Variable

11.4.6 pthread_cond_destroy - 销毁条件变量

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_cond_destroy(
   pthread_cond_t *cond
);
```

STATUS CODES: 状态码：

EINVAL The specified condition variable is invalid.
EINVAL 指定的条件变量无效。

EBUSY The specified condition variable is currently in use.
EBUSY 指定的条件变量当前正在使用中。

DESCRIPTION: 描述：

NOTES: 注意事项：

11.4.7 pthread_cond_signal - Signal a Condition Variable

11.4.7 pthread_cond_signal - 用信号发送条件变量

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_cond_signal(
   pthread_cond_t *cond
);
```

STATUS CODES: 状态码：

EINVAL	The specified condition variable is not valid.
EINVAL	指定的条件变量无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine should not be invoked from a handler from an asynchronous signal handler or an interrupt service routine.
这个例程不应从来自异步信号处理程序的处理程序或中断服务例程调用。

11.4.8 pthread_cond_broadcast - Broadcast a Condition Variable

11.4.8 pthread_cond_broadcast - 广播条件变量

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_cond_broadcast(
   pthread_cond_t *cond
);
```

STATUS CODES: 状态码：

EINVAL	The specified condition variable is not valid.
EINVAL	指定的条件变量无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

This routine should not be invoked from a handler from an asynchronous signal handler or an interrupt service routine.
这个例程不应从来自异步信号处理程序的处理程序或中断服务例程调用。

11.4.9 pthread_cond_wait - Wait on a Condition Variable

11.4.9 pthread_cond_wait - 等待条件变量

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_cond_wait(
   pthread_cond_t    *cond,
   pthread_mutex_t *mutex
);
```

STATUS CODES: 状态码：

EINVAL The specified condition variable or mutex is not initialized OR different mutexes were specified for concurrent pthread_cond_wait() and pthread_cond_timedwait() operations on the same condition variable OR the mutex was not owned by the current thread at the time of the call.
EINVAL 未初始化指定的条件变量或互斥体，或为并发的 pthread_cond_wait() 指定了不同的互斥体，并且在调用时在同一条件变量或互斥体上的 pthread_cond_timedwait()操作不由当前线程拥有。

DESCRIPTION: 描述：

NOTES: 注意事项：

11.4.10 pthread_cond_timedwait - Wait with Timeout a Condition Variable

11.4.10 pthread_cond_timedwait - 超时等待条件变量

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_cond_timedwait(
   pthread_cond_t    *cond,
   pthread_mutex_t    *mutex,
   const struct timespec *abstime
);
```

STATUS CODES: 状态码：

EINVAL The specified condition variable or mutex is not initialized OR different mutexes were specified for concurrent pthread_cond_wait() and pthread_cond_timedwait() operations on the same condition variable OR the mutex was not owned by the current thread at the time of the call.
EINVAL 未初始化指定的条件变量或互斥体，或为并发的 pthread_cond_wait() 指定了不同的互斥体，并且在调用时在同一条件变量或互斥体上的 pthread_cond_timedwait()操作不由当前线程拥有。

ETIMEDOUT The specified time has elapsed without the condition variable being satisfied.
ETIMEDOUT 指定的时间已流逝，没有条件变量被满足。

DESCRIPTION: 描述：

NOTES: 注意事项：

12 Memory Management Manager

12 内存管理管理器

12.1 Introduction

12.1 简介

The memory management manager is ...
内存管理管理器是……

The directives provided by the memory management manager are:
由内存管理管理器提供的指令有：

- mlockall - Lock the Address Space of a Process
- munlockall - Unlock the Address Space of a Process
- mlock - Lock a Range of the Process Address Space
- munlock - Unlock a Range of the Process Address Space
- mmap - Map Process Addresses to a Memory Object
- munmap - Unmap Previously Mapped Addresses
- mprotect - Change Memory Protection
- msync - Memory Object Synchronization
- shm_open - Open a Shared Memory Object
- shm_unlink - Remove a Shared Memory Object

- mlockall - 锁定进程的地址空间
- munlockall - 解锁进程的地址空间
- mlock - 锁定一系列进程地址空间
- munlock - 解锁一系列进程地址空间
- mmap - 映射进程地址到内存对象
- munmap - 取消映射先前映射的地址
- mprotect - 更改内存保护
- msync - 内存对象同步
- shm_open - 打开共享的内存对象
- shm_unlink - 移除共享的内存对象

12.2 Background

12.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

12.3 Operations

12.3 操作

There is currently no text in this section.
当前在本节中没有文本。

12.4 Directives

12.4 指令

This section details the memory management manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了内存管理管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

12.4.1 mlockall - Lock the Address Space of a Process

12.4.1 mlockall - 锁定进程的地址空间

CALLING SEQUENCE: 调用序列：

```
int mlockall(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.2 munlockall - Unlock the Address Space of a Process

12.4.2 munlockall - 解锁进程的地址空间

CALLING SEQUENCE: 调用序列：

```
int munlockall(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.3 mlock - Lock a Range of the Process Address Space

12.4.3 mlock - 锁定一系列进程地址空间

CALLING SEQUENCE: **调用序列：**

```
int mlock(
);
```

STATUS CODES: **状态码：**

E The

DESCRIPTION: **描述：**

NOTES: **注意事项：**

12.4.4 munlock - Unlock a Range of the Process Address Space

12.4.4 munlock - 解锁一系列进程地址空间

CALLING SEQUENCE: 调用序列：

```
int munlock(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.5 mmap - Map Process Addresses to a Memory Object

12.4.5 mmap - 映射进程地址到内存对象

CALLING SEQUENCE: 调用序列：

```
int mmap(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.6 munmap - Unmap Previously Mapped Addresses

12.4.6 munmap - 取消映射先前映射的地址

CALLING SEQUENCE: 调用序列：

```
int munmap(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.7 mprotect - Change Memory Protection

12.4.7 mprotect - 更改内存保护

CALLING SEQUENCE: 调用序列：

```
int mprotect(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.8 msync - Memory Object Synchronization

12.4.8 msync - 内存对象同步

CALLING SEQUENCE: 调用序列：

```
int msync(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.9 shm_open - Open a Shared Memory Object

12.4.9 shm_open - 打开共享的内存对象

CALLING SEQUENCE: 调用序列：

```
int shm_open(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

12.4.10 shm_unlink - Remove a Shared Memory Object

12.4.10 shm_unlink - 移除共享的内存对象

CALLING SEQUENCE: 调用序列：

```
int shm_unlink(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

13 Scheduler Manager

13 调度器管理器

13.1 Introduction

13.1 简介

The scheduler manager ...
调度器管理器……

The directives provided by the scheduler manager are:
由调度器管理器提供的指令有：

- sched_get_priority_min - Get Minimum Priority Value
- sched_get_priority_max - Get Maximum Priority Value
- sched_rr_get_interval - Get Timeslicing Quantum
- sched_yield - Yield the Processor

- sched_get_priority_min - 获得最低优先级值
- sched_get_priority_max - 获得最高优先级值
- sched_rr_get_interval - 获得时间片轮转量子
- sched_yield - 让出处理器

13.2 Background

13.2 背景

13.2.1 Priority

13.2.1 优先级

In the RTEMS implementation of the POSIX API, the priorities range from the low priority of sched_get_priority_min() to the highest priority of sched_get_priority_max(). Numerically higher values represent higher priorities.
在 POSIX API 的 RTEMS 实现中，优先级范围从 sched_get_priority_min()的低优先级到 sched_get_priority_max()的高优先级。数值较高的值代表较高的优先级。

13.2.2 Scheduling Policies

13.2.2 调度策略

The following scheduling policies are available:
下列调度策略是可用的：

SCHED_FIFO	Priority-based, preemptive scheduling with no timeslicing. This is equivalent to what is called "manual round-robin" scheduling.
SCHED_FIFO	没有时间片轮转的基于优先级的抢占式调度。这相当于所谓的“手动轮转”调度。
SCHED_RR	Priority-based, preemptive scheduling with timeslicing. Time quantums are maintained on a per-thread basis and are not reset at each context switch. Thus, a thread which is preempted and subsequently resumes execution will attempt to complete the unused portion of its time quantum.
SCHED_RR	具有时间片轮转的基于优先级的抢占式调度。基于每线程维护时间量子，不会在每次上下文切换时复位。因此，被抢占并随后恢复执行的线程将尝试完成其时间量子的未使用部分。
SCHED_OTHER	Priority-based, preemptive scheduling with timeslicing. Time quantums are maintained on a per-thread basis and are reset at each context switch.
SCHED_OTHER	具有时间片轮转的基于优先级的抢占式调度。基于每线程维护时间量子，在每次上下文切换时复位。
SCHED_SPORADIC	Priority-based, preemptive scheduling utilizing three additional parameters: budget, replenishment period, and low priority. Under this policy, the thread is allowed to execute for "budget" amount of time before its priority is lowered to "low priority". At the end of each replenishment period, the thread resumes its initial priority and has its budget replenished.
SCHED_SPORADIC	基于优先级的抢占式调度，利用三个附加参数：预算、补充周期和低优先级。在此策略下，允许线程在其优先级被降为“低优先级”之前执行“预算”数量的时间。在每个补充周期的末尾，线程恢复其初始优先级，并有其补充的预算。

13.3 Operations

13.3 操作

There is currently no text in this section.
当前在本节中没有文本。

13.4 Directives

13.4 指令

This section details the scheduler manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了调度器管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

13.4.1 schedget_priority_min - Get Minimum Priority Value

13.4.1 schedget_priority_min - 获得最低优先级值

CALLING SEQUENCE: 调用序列：

```
#include <sched.h>

int sched_get_priority_min(
   int policy
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为下列值之一：

EINVAL	The indicated policy is invalid.
EINVAL	表示的策略无效。

DESCRIPTION: 描述：

This routine return the minimum (numerically and logically lowest) priority for the specified policy.
这个例程返回指定 policy 的最低（数值上和逻辑上最低）优先级。

NOTES: 注意事项：

NONE
无

13.4.2 sched_get_priority_max - Get Maximum Priority Value

13.4.2 sched_get_priority_max - 获得最高优先级值

CALLING SEQUENCE: 调用序列：

```
#include <sched.h>

int sched_get_priority_max(
   int policy
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为下列值之一：

EINVAL	The indicated policy is invalid.
EINVAL	表示的策略无效。

DESCRIPTION: 描述：

This routine return the maximum (numerically and logically highest) priority for the specified policy.
这个例程返回指定 policy 的最高（数值上和逻辑上最高）优先级。

NOTES: 注意事项：

NONE
无

13.4.3 sched_rr_get_interval - Get Timeslicing Quantum

13.4.3 sched_rr_get_interval - 获得时间片轮转量子

CALLING SEQUENCE: 调用序列：

```
#include <sched.h>

int sched_rr_get_interval(
    pid_t    pid,
    struct timespec *interval
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为下列值之一：

ESRCH The indicated process id is invalid.
ESRCH 表示的进程 id 无效。

EINVAL The specified interval pointer parameter is invalid.
EINVAL 指定的间隔指针参数无效。

DESCRIPTION: 描述：

This routine returns the length of the timeslice quantum in the interval parameter for the specified pid.
这个例程在 interval 参数中为指定的 pid 返回时间片量子的长度。

NOTES: 注意事项：

The pid argument should be 0 to indicate the calling process.
pid 参数应为 0 以表示调用进程。

13.4.4 sched_yield - Yield the Processor

13.4.4 sched_yield - 让出处理器

CALLING SEQUENCE: 调用序列：

```
#include <sched.h>

int sched_yield( void );
```

STATUS CODES: 状态码：

This routine always returns zero to indicate success.
这个例程总是返回 0 以表示成功。

DESCRIPTION: 描述：

This call forces the calling thread to yield the processor to another thread. Normally this is used to implement voluntary round-robin task scheduling.
这个调用强迫调用线程让出处理器给另一个线程。通常这用于实现自动轮转任务调度。

NOTES: 注意事项：

NONE
无

14 Clock Manager

14 时钟管理器

14.1 Introduction

14.1 简介

The clock manager provides services two primary classes of services. The first focuses on obtaining and setting the current date and time. The other category of services focus on allowing a thread to delay for a specific length of time.
时钟管理器提供两个主要类别的服务。第一类侧重于获取和设置当前日期和时间。其他类别的服务侧重于允许线程延时特定长度的时间。

The directives provided by the clock manager are:
时钟管理器提供的指令有：

- clock_gettime - Obtain Time of Day
- clock_settime - Set Time of Day
- clock_getres - Get Clock Resolution
- sleep - Delay Process Execution
- usleep - Delay Process Execution in Microseconds
- nanosleep - Delay with High Resolution
- gettimeofday - Get the Time of Day
- time - Get time in seconds

- clock_gettime - 获取日历时间
- clock_settime - 设置日历时间
- clock_getres - 获得时钟分辨率
- sleep - 延迟进程执行
- usleep - 以微秒为单位延迟进程执行
- nanosleep - 具有高分辨率的延迟
- gettimeofday - 获得日历时间
- time - 获得以秒为单位的时间

14.2 Background

14.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

14.3 Operations

14.3 操作

There is currently no text in this section.
当前在本节中没有文本。

14.4 Directives

14.4 指令

This section details the clock manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了时钟管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

14.4.1 clock_gettime - Obtain Time of Day

14.4.1 clock_gettime - 获取日历时间

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int clock_gettime(
   clockid_t   clock_id,
   struct timespec *tp
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为下列值之一：

EINVAL	The tp pointer parameter is invalid.
EINVAL	tp 指针参数无效。
EINVAL	The clock_id specified is invalid.
EINVAL	指定的 clock_id 无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

NONE
无

14.4.2 clock_settime - Set Time of Day

14.4.2 clock_settime - 设置日历时间

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int clock_settime(
   clockid_t    clock_id,
   const struct timespec *tp
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为下列值之一：

EINVAL The tp pointer parameter is invalid.
EINVAL tp 指针参数无效。

EINVAL The clock_id specified is invalid.
EINVAL 指定的 clock_id 无效。

EINVAL The contents of the tp structure are invalid.
EINVAL tp 结构的内容无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

NONE
无

14.4.3 clock_getres - Get Clock Resolution

14.4.3 clock_getres - 获得时钟分辨率

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int clock_getres(
   clockid_t    clock_id,
   struct timespec *res
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为下列值之一：

EINVAL The res pointer parameter is invalid.
EINVAL res 指针参数无效。

EINVAL The clock_id specified is invalid.
EINVAL 指定的 clock_id 无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

If res is NULL, then the resolution is not returned.
如果 res 为 NULL，则不返回分辨率。

14.4.4 sleep - Delay Process Execution

14.4.4 sleep - 延迟进程执行

CALLING SEQUENCE: 调用序列：

```
#include <unistd.h>

unsigned int sleep(
   unsigned int seconds
);
```

STATUS CODES: 状态码：

This routine returns the number of unslept seconds.
这个例程返回未休眠的秒数。

DESCRIPTION: 描述：

The sleep() function delays the calling thread by the specified number of seconds.
sleep()函数将调用线程延迟指定的秒数。

NOTES: 注意事项：

This call is interruptible by a signal.
这个调用可被一个信号中断。

14.4.5 usleep - Delay Process Execution in Microseconds

14.4.5 usleep - 以微秒为单位延迟进程执行

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

useconds_t      usleep(
   useconds_t useconds
);
```

STATUS CODES: 状态码：

This routine returns the number of unslept seconds.
这个例程返回未休眠的秒数。

DESCRIPTION: 描述：

The sleep() function delays the calling thread by the specified number of seconds.
sleep()函数将调用线程延迟指定的秒数。

The usleep() function suspends the calling thread from execution until either the number of microseconds specified by the useconds argument has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal-catching function or to terminate the process.
usleep()函数从执行挂起调用线程，直到由 useconds 参数指定的微秒数已流逝，或一个信号被投递到调用线程，并且其操作是调用信号捕捉函数或终止该进程。

Because of other activity, or because of the time spent in processing the call, the actual length of time the thread is blocked may be longer than the amount of time specified.
由于其他操作，或由于时间花在处理该调用上，线程被阻塞的时间的实际长度可能超过指定的时间数量。

NOTES: 注意事项：

This call is interruptible by a signal.
这个调用可被一个信号中断。

The Single UNIX Specification allows this service to be implemented using the same timer as that used by the alarm() service. This is **NOT** the case for **RTEMS** and this call has no interaction with the SIGALRM signal.
Single UNIX Specification 允许此服务使用由 alarm()服务使用的相同定时器实现。这不是 RTEMS 的情况，此调用没有与 SIGALRM 信号的交互。

14.4.6 nanosleep - Delay with High Resolution

14.4.6 nanosleep - 具有高分辨率的延迟

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int nanosleep(
   const struct timespec *rqtp,
   struct timespec    *rmtp
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno to one of the following:
出现错误时，此例程返回-1 并设置 errno 为下列值之一：

EINTR The routine was interrupted by a signal.
EINTR 例程被一个信号中断。

EAGAIN The requested sleep period specified negative seconds or nanoseconds.
EAGAIN 请求的休眠周期指定了负秒或纳秒。

EINVAL The requested sleep period specified an invalid number for the nanoseconds field.
EINVAL 请求的休眠周期指定了纳秒字段的无效数字。

DESCRIPTION: 描述：

NOTES: 注意事项：

This call is interruptible by a signal.
这个调用可被一个信号中断。

14.4.7 gettimeofday - Get the Time of Day

14.4.7 gettimeofday - 获得日历时间

CALLING SEQUENCE: 调用序列：

```
#include <sys/time.h>
#include <unistd.h>

int gettimeofday(
   struct timeval    *tp,
   struct timezone *tzp
);
```

STATUS CODES: 状态码：

On error, this routine returns -1 and sets errno as appropriate.
出现错误时，此例程返回-1 并酌情设置 errno。

EPERM settimeofdat is called by someone other than the superuser.
EPERM settimeofdat 由超级用户之外的人调用。

EINVAL Timezone (or something else) is invalid.
EINVAL 时区（或别的东西）无效。

EFAULT One of tv or tz pointed outside your accessible address space
EFAULT tv 或 tz 指向你可访问的地址空间之外。

DESCRIPTION: 描述：

This routine returns the current time of day in the tp structure.
这个例程在 tp 结构中返回当前的日历时间。

NOTES: 注意事项：

Currently, the timezone information is not supported. The tzp argument is ignored.
当前，不支持时区信息。tzp 参数被忽略。

14.4.8 time - Get time in seconds

14.4.8 time - 获得以秒为单位的时间

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int time(
   time_t *tloc
);
```

STATUS CODES: 状态码：

This routine returns the number of seconds since the Epoch.
这个例程返回自新纪元以来的秒数。

DESCRIPTION: 描述：

time returns the time since 00:00:00 GMT, January 1, 1970, measured in seconds
time 返回以秒测量的自 1970 年 1 月 1 日 00:00:00 GMT 以来的时间。

If tloc in non null, the return value is also stored in the memory pointed to by t.
如果 tloc 不是 null，返回值还被存储在由 t 指向的内存中。

NOTES: 注意事项：

NONE
无

15 Timer Manager

15 定时器管理器

15.1 Introduction

15.1 简介

The timer manager is ...
定时器管理器是……

The services provided by the timer manager are:
由定时器管理器提供的服务有：

- timer_create - Create a Per-Process Timer
- timer_delete - Delete a Per-Process Timer
- timer_settime - Set Next Timer Expiration
- timer_gettime - Get Time Remaining on Timer
- timer_getoverrun - Get Timer Overrun Count

- timer_create - 创建每进程定时器
- timer_delete - 删除每进程定时器
- timer_settime - 设置下一个定时器到期时间
- timer_gettime - 获得定时器上的剩余时间
- timer_getoverrun - 获得定时器溢出计数

15.2 Background

15.2 背景知识

15.3 Operations

15.3 操作

15.4 System Calls

15.4 系统调用

This section details the timer manager’s services. A subsection is dedicated to each of this manager’s services and describes the calling sequence, related constants, usage, and status codes.

本节详细介绍了定时器管理器的服务。一小节专注于该管理器的一个服务，描述了调用序列、相关的常量、用法和状态码。

15.4.1 timer_create - Create a Per-Process Timer

15.4.1 timer_create - 创建每进程定时器

CALLING SEQUENCE: 调用序列：

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

int timer_create(
   clockid_t        clock_id,
   struct sigevent *evp,
   timer_t    *timerid
);
```

STATUS CODES: 状态码：

EXXX -

DESCRIPTION: 描述：

NOTES: 注意事项：

15.4.2 timer_delete - Delete a Per-Process Timer

15.4.2 timer_delete - 删除每进程定时器

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int timer_delete(
   timer_t timerid
);
```

STATUS CODES: 状态码：

EXXX -

DESCRIPTION: 描述：

NOTES: 注意事项：

15.4.3 timer_settime - Set Next Timer Expiration

15.4.3 timer_settime - 设置下一个定时器到期时间

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int timer_settime(
    timer_t    timerid,
    int    flags,
    const struct itimerspec *value,
    struct itimerspec    *ovalue
);
```

STATUS CODES: 状态码：

EXXX -

DESCRIPTION: 描述：

NOTES: 注意事项：

15.4.4 timer_gettime - Get Time Remaining on Timer

15.4.4 timer_gettime - 获得定时器上的剩余时间

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int timer_gettime(
   timer_t    timerid,
   struct itimerspec *value
);
```

STATUS CODES: 状态码：

EXXX -

DESCRIPTION: 描述：

NOTES: 注意事项：

15.4.5 timer_getoverrun - Get Timer Overrun Count

15.4.5 timer_getoverrun - 获得定时器溢出计数

CALLING SEQUENCE: 调用序列：

```
#include <time.h>

int timer_getoverrun(
   timer_t    timerid
);
```

STATUS CODES: 状态码：

EXXX -

DESCRIPTION: 描述：

NOTES: 注意事项：

16 Message Passing Manager

16 消息传递管理器

16.1 Introduction

16.1 简介

The message passing manager is the means to provide communication and synchronization capabilities using POSIX message queues.
消息传递管理器打算使用 POSIX 消息队列提供通信同步功能。

The directives provided by the message passing manager are:
由消息传递管理器提供的指令有：

- mq_open - Open a Message Queue
- mq_close - Close a Message Queue
- mq_unlink - Remove a Message Queue
- mq_send - Send a Message to a Message Queue
- mq_receive - Receive a Message from a Message Queue
- mq_notify - Notify Process that a Message is Available
- mq_setattr - Set Message Queue Attributes
- mq_getattr - Get Message Queue Attributes

- mq_open - 打开消息队列
- mq_close - 关闭消息队列
- mq_unlink - 移除消息队列
- mq_send - 发送消息到消息队列
- mq_receive - 从消息队列接收消息
- mq_notify - 通知进程消息可用
- mq_setattr - 设置消息队列属性
- mq_getattr - 获得消息队列属性

16.2 Background

16.2 背景知识

16.2.1 Theory

16.2.1 理论

Message queues are named objects that operate with readers and writers. In addition, a message queue is a priority queue of discrete messages. POSIX message queues offer a certain, basic amount of application access to, and control over, the message queue geometry that can be changed.
消息队列是与读取者和写入者一起操作的命名对象。此外，消息队列是离散消息的优先级队列。POSIX 消息队列提供一定的、基本的应用程序访问，和可以更改的消息队列几何的控制权。

16.2.2 Messages

16.2.2 消息

A message is a variable length buffer where information can be stored to support communication. The length of the message and the information stored in that message are user-defined and can be actual data, pointer(s), or empty. There is a maximum acceptable length for a message that is associated with each message queue.
消息是可变长度的缓冲区，可以存储信息以支持通信。消息的长度和在该消息中存储的信息是用户定义的，可以为实际数据、指针或空。消息的最大可接受长度是与每个消息队列相关联的。

16.2.3 Message Queues

16.2.3 消息队列

Message queues are named objects similar to the pipes of POSIX. They are a means of communicating data between multiple processes and for passing messages among tasks and ISRs. Message queues can contain a variable number of messages from 0 to an upper limit that is user defined. The maximum length of the message can be set on a per message queue basis. Normally messages are sent and received from the message queue in FIFO order. However, messages can also be prioritized and a priority queue established for the passing of messages. Synchronization is needed when a task waits for a message to arrive at a queue. Also, a task may poll a queue for the arrival of a message.
消息队列是类似于 POSIX 管道的命名对象。它们是在多个进程之间交流数据，和在任务和 ISR 中间传递消息的方法。消息队列可以包含可变数量的消息，从 0 到

用户定义的上限。消息的最大长度可以基于每个消息队列设置。通常以 FIFO 顺序从消息队列发送和接收消息。然而，消息也可以被优先级化，是一个为消息传递建立的优先级队列。当任务等待消息到达队列时需要同步。此外，任务可以为消息的到达轮询一个队列。

The message queue descriptor mqd_t represents the message queue. It is passed as an argument to all of the message queue functions.
消息队列描述符 mqd_t 代表消息队列。它被作为参数传递给所有消息队列函数。

16.2.4 Building a Message Queue Attribute Set

16.2.4 建立消息队列属性集

The mq_attr structure is used to define the characteristics of the message queue.
mq_attr 结构用于定义消息队列的特性。

```
typedef    struct    mq_attr{
   long mq_flags;
   long mq_maxmsg;
   long mq_msgsize;
   long mq_curmsgs;
};
```

All of these attributes are set when the message queue is created using mq_open. The mq_flags field is not used in the creation of a message queue, it is only used by mq_setattr and mq_getattr. The structure mq_attr is passed as an argument to mq_setattr and mq_getattr.
当使用 mq_open 创建消息队列时设置所有这些属性。在创建消息队列时不使用 mq_flags 字段，它只由 mq_setattr 和 mq_getattr 使用。结构 mq_attr 被作为参数传递给 mq_setattr 和 mq_getattr。

The mq_flags contain information affecting the behavior of the message queue. The O_NONBLOCK mq_flag is the only flag that is defined. In mq_setattr, the mq_flag can be set to dynamically change the blocking and non-blocking behavior of the message queue. If the non-block flag is set then the message queue is non-blocking, and requests to send and receive messages do not block waiting for resources. For a blocking message queue, a request to send might have to wait for an empty message queue, and a request to receive might have to wait for a message to arrive on the queue. Both mq_maxmsg and mq_msgsize affect the sizing of the message queue.

mq_maxmsg specifies how many messages the queue can hold at any one time. mq_msgsize specifies the size of any one message on the queue. If either of these limits is exceeded, an error message results.
mq_flags 包含影响消息队列行为的信息。O_NONBLOCK mq_flag 是被定义的唯一标志。在 mq_setattr 中，mq_flag 可以被设置为动态更改消息队列的阻塞和非阻塞行为。如果设置了非阻塞标志，则消息队列是非阻塞的，发送和接收消息的请求不会阻塞等待的资源。对于阻塞的消息队列，发送的消息可能必须等待空消息队列，接收的请求可能必须等待消息到达队列。mq_maxmsg 和 mq_msgsize 影响消息队列的大小调整。mq_maxmsg 指定队列在任何一个时间可以持有多少个消息。mq_msgsize 指定队列上的任何一个消息的大小。如果超过这些限制之一，则产生错误消息。

Upon return from mq_getattr, the mq_curmsgs is set according to the current state of the message queue. This specifies the number of messages currently on the queue.
从 mq_getattr 返回之后，根据消息队列的当前状态设置 mq_curmsgs。这将指定当前在队列上的消息数量。

16.2.5 Notification of a Message on the Queue

16.2.5 队列上的消息通知

Every message queue has the ability to notify one (and only one) process whenever the queue’s state changes from empty (0 messages) to nonempty. This means that the process does not have to block or constantly poll while it waits for a message. By calling mq_notify, you can attach a notification request to a message queue. When a message is received by an empty queue, if there are no processes blocked and waiting for the message, then the queue notifies the requesting process of a message arrival. There is only one signal sent by the message queue, after that the notification request is de-registered and another process can attach its notification request. After receipt of a notification, a process must re-register if it wishes to be notified again.
每当队列状态从空（0 消息）变为非空时，每个消息队列都有通知一个（且只有一个）进程的能力。这意味着进程不必在等待消息时阻塞或不断地轮询。通过调用 mq_notify，你可以连接通知请求到消息队列，当消息由空队列接收时，如果没有进程阻塞并等待消息，则队列通知消息到达的请求进程。只有一个信号通过消息队列发送，然后通知请求被取消注册，另一个进程可以连接其通知请求。在收到通知之后，如果进程希望再次被通知，它必须重新注册。

If there is a process blocked and waiting for the message, that process gets the message, and notification is not sent. It is also possible for another process to receive the message after the notification is sent but before the notified process has sent its receive request.
如果有进程阻塞并等待消息，该进程获得消息，并且不发送通知。还有可能是，在发送通知之后，但在通知的进程已发送其接收请求之前，另一个进程接收了该消息。

Only one process can have a notification request attached to a message queue at any one time. If another process attempts to register a notification request, it fails. You can de-register for a message queue by passing a NULL to mq_notify, this removes any notification request attached to the queue. Whenever the message queue is closed, all notification attachments are removed.
在任何一个时间，只有一个进程可以有通知请求连接到消息队列。如果另一个进程试图注册通知请求，它将失败。你可以通过传递 NULL 到 mq_notify，取消注册消息队列，这将移除连接到该队列的所有通知请求。每当关闭消息队列时，移除所有通知连接。

16.2.6 POSIX Interpretation Issues

16.2.6 POSIX 解释问题

There is one significant point of interpretation related to the RTEMS implementation of POSIX message queues:
解释有关 POSIX 消息队列的 RTEMS 实现的重要一点是：

What happens to threads already blocked on a message queue when the mode of that same message queue is changed from blocking to non-blocking?
当相同消息队列的模式从阻塞变成非阻塞时，已阻塞在消息队列上的线程会发生什么？

The RTEMS POSIX implementation decided to unblock all waiting tasks with an EAGAIN status just as if a non-blocking version of the same operation had returned unsatisfied. This case is not discussed in the POSIX standard and other implementations may have chosen alternative behaviors.
RTEMS POSIX 实现解决了解锁就像非阻塞版本的相同操作返回了不满足的具有 EAGAIN 状态的所有等待任务。在 POSIX 标准中不讨论这种情况，其他实现可能选择了替代行为。

16.3 Operations

16.3 操作

16.3.1 Opening or Creating a Message Queue

16.3.1 打开或创建消息队列

If the message queue already exists, mq_open() opens it, if the message queue does not exist, mq_open() creates it. When a message queue is created, the geometry of the message queue is contained in the attribute structure that is passed in as an argument. This includes mq_msgsize that dictates the maximum size of a single message, and the mq_maxmsg that dictates the maximum number of messages the queue can hold at one time. The blocking or non-blocking behavior of the queue can also specified.
如果消息队列已存在，mq_open()打开它，如果消息队列不存在，mq_open()创建它。当创建消息队列时，消息队列的几何被包含在作为参数传入的属性结构中。这包括表示单个消息的最大大小的 mq_msgsize，和表示队列可以同时持有的消息的最大数量的 mq_maxmsg。还可以指定队列的阻塞或非阻塞行为。

16.3.2 Closing a Message Queue

16.3.2 关闭消息队列

The mq_close() function is used to close the connection made to a message queue that was made during mq_open. The message queue itself and the messages on the queue are persistent and remain after the queue is closed.
mq_close()函数用于关闭在 mq_open 时建立的消息队列的连接。消息队列本身和在队列上的消息是持续存在的，在关闭队列之后保留。

16.3.3 Removing a Message Queue

16.3.3 移除消息队列

The mq_unlink() function removes the named message queue. If the message queue is not open when mq_unlink is called, then the queue is immediately eliminated. Any messages that were on the queue are lost, and the queue can not be opened again. If processes have the queue open when mq_unlink is called, the removal of the queue is delayed until the last process using the queue has finished. However, the name of the message queue is removed so that no other process can open it.

mq_unlink()函数移除指定名称的消息队列。如果当调用 mq_unlink 时消息队列未打开，则该队列被立即排除。在队列上的任何消息都将丢失，并且不能再次打开该队列。如果当调用 mq_unlink 时进程有打开的队列，队列的移除将延迟，直到使用该队列的最后一个进程已完成。然而，该消息队列的名称被移除，所以没有其他进程可以打开它。

16.3.4 Sending a Message to a Message Queue

16.3.4 发送消息到消息队列

The mq_send() function adds the message in priority order to the message queue. Each message has an assigned a priority. The highest priority message is be at the front of the queue.
mq_send()函数以优先级顺序添加消息到消息队列。每个消息都有分配的优先级。最高优先级的消息处于队列的前面。

The maximum number of messages that a message queue may accept is specified at creation by the mq_maxmsg field of the attribute structure. If this amount is exceeded, the behavior of the process is determined according to what oflag was used when the message queue was opened. If the queue was opened with O_NONBLOCK flag set, the process does not block, and an error is returned. If the O_NONBLOCK flag was not set, the process does block and wait for space on the queue.
消息队列可以接受的消息的最大数量是在创建的属性结构的 mq_maxmsg 字段上指定的。如果超过了此数量，进程的行为根据在打开消息队列时使用的什么样的 oflag 确定的。如果使用 O_NONBLOCK 标志设置打开队列，进程不阻塞，并返回错误。如果未设置 O_NONBLOCK 标志，进程阻塞，并等待队列上的空间。

16.3.5 Receiving a Message from a Message Queue

16.3.5 从消息队列接收消息

The mq_receive() function is used to receive the oldest of the highest priority message(s) from the message queue specified by mqdes. The messages are received in FIFO order within the priorities. The received message’s priority is stored in the location referenced by the msg_prio. If the msg_prio is a NULL, the priority is discarded. The message is removed and stored in an area pointed to by msg_ptr whose length is of msg_len. The msg_len must be at least equal to the mq_msgsize attribute of the message queue.
mq_receive()函数用于从 mqdes 所指定的消息队列接收最旧的最高优先级的消息。

在优先级内以 FIFO 顺序接收消息。在 msg_prio 所引用的位置中存储接收到的消息的优先级。如果 msg_prio 为 NULL，则丢弃该优先级。该消息被移除，并被存储在长度为 msg_len 的由 msg_ptr 指向的区域中。msg_len 必须至少等于消息队列的 mq_msgsize 属性。

The blocking behavior of the message queue is set by O_NONBLOCK at mq_open or by setting O_NONBLOCK in mq_flags in a call to mq_setattr. If this is a blocking queue, the process does block and wait on an empty queue. If this a non-blocking queue, the process does not block. Upon successful completion, mq_receive returns the length of the selected message in bytes and the message is removed from the queue.
消息队列的阻塞行为是由 mq_open 中的 O_NONBLOCK 或由调用 mq_setattr 时的 mq_flags 中的设置 O_NONBLOCK 设置的。如果这是阻塞队列，进程阻塞并等待在空队列。如果这是非阻塞队列，进程不阻塞。成功完成之后，mq_receive 返回以字节为单位的所选消息的长度，并从队列移除该消息。

16.3.6 Notification of Receipt of a Message on an Empty Queue

16.3.6 在空队列上收到消息的通知

The mq_notify() function registers the calling process to be notified of message arrival at an empty message queue. Every message queue has the ability to notify one (and only one) process whenever the queue’s state changes from empty (0 messages) to nonempty. This means that the process does not have to block or constantly poll while it waits for a message. By calling mq_notify, a notification request is attached to a message queue. When a message is received by an empty queue, if there are no processes blocked and waiting for the message, then the queue notifies the requesting process of a message arrival. There is only one signal sent by the message queue, after that the notification request is de-registered and another process can attach its notification request. After receipt of a notification, a process must re-register if it wishes to be notified again.
mq_notify()函数注册调用进程，以通知消息到达了空消息队列。每当队列状态从空（0 消息）变为非空时，每个消息队列都有通知一个（且只有一个）进程的能力。这意味着进程不必在等待消息时阻塞或不断地轮询。通过调用 mq_notify，你可以连接通知请求到消息队列，当消息由空队列接收时，如果没有进程阻塞并等待消息，则队列通知消息到达的请求进程。只有一个信号通过消息队列发送，然后通知请求被取消注册，另一个进程可以连接其通知请求。在收到通知之后，如果进程希望再次被通知，它必须重新注册。

If there is a process blocked and waiting for the message, that process gets the

message, and notification is not sent. Only one process can have a notification request attached to a message queue at any one time. If another process attempts to register a notification request, it fails. You can de-register for a message queue by passing a NULL to mq_notify, this removes any notification request attached to the queue. Whenever the message queue is closed, all notification attachments are removed.
如果有进程阻塞并等待消息，该进程获得消息，并且不发送通知。在任何一个时间，只有一个进程可以有通知请求连接到消息队列。如果另一个进程试图注册通知请求，它将失败。你可以通过传递 NULL 到 mq_notify，取消注册消息队列，这将移除连接到该队列的所有通知请求。每当关闭消息队列时，移除所有通知连接。

16.3.7 Setting the Attributes of a Message Queue

16.3.7 设置消息队列属性

The mq_setattr() function is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by mqdes. The *omqstat represents the old or previous attributes. If omqstat is non-NULL, the function mq_setattr() stores, in the location referenced by omqstat, the previous message queue attributes and the current queue status. These values are the same as would be returned by a call to mq_getattr() at that point.
mq_setattr()函数用于设置与 mqdes 所指定的消息队列描述符所引用的打开消息队列描述相关联的属性。*omqstat 代表早的或先前的属性。如果 omqstat 为非 NULL，函数 mq_setattr()存储在 omqstat、先前的消息队列属性和当前队列的状态所引用的位置。这些值同样是通过在此时调用 mq_getattr()返回。

There is only one mq_attr.mq_flag that can be altered by this call. This is the flag that deals with the blocking and non-blocking behavior of the message queue. If the flag is set then the message queue is non-blocking, and requests to send or receive do not block while waiting for resources. If the flag is not set, then message send and receive may involve waiting for an empty queue or waiting for a message to arrive.
只有一个可以由此调用改变的 mq_attr.mq_flag。这是处理消息队列的阻塞和非阻塞行为的标志。如果设置了该标志，则消息队列是非阻塞的，当等待资源时，发送或接收的请求不会阻塞。如果未设置该标志，则消息发送和接收可能涉及等待空队列或等待消息的到达。

16.3.8 Getting the Attributes of a Message Queue

16.3.8 获得消息队列属性

The mq_getattr() function is used to get status information and attributes of the message queue associated with the message queue descriptor. The results are returned in the mq_attr structure referenced by the mqstat argument. All of these attributes are set at create time, except the blocking/non-blocking behavior of the message queue which can be dynamically set by using mq_setattr. The attribute mq_curmsg is set to reflect the number of messages on the queue at the time that mq_getattr was called.
mq_getattr()函数用于获得与消息队列描述符相关联的消息队列的状态信息和属性。在 mqstat 参数所引用的 mq_attr 结构中返回结果。所有这些属性在创建时被设置，除了可以通过使用 mq_setattr 动态设置的消息队列的阻塞/非阻塞行为。属性 mq_curmsg 被设置，以反映在调用 mq_getattr 时在队列上的消息的数量。

16.4 Directives

16.4 指令

This section details the message passing manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了消息传递管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

16.4.1 mq_open - Open a Message Queue

16.4.1 mq_open - 打开消息队列

CALLING SEQUENCE: 调用序列：

```
#include <mqueue.h>

mqd_t mq_open(
   const char    *name,
   int    oflag,
   mode_t    mode,
   struct mq_attr *attr
);
```

STATUS CODES: 状态码：

EACCES - Either the message queue exists and the permissions requested in oflags were denied, or the message does not exist and permission to create one is denied.
EACCES - 消息队列存在，并且在 oflags 中请求的权限被拒绝，或消息不存在，并且创建一个消息的权限被拒绝。

EEXIST - You tried to create a message queue that already exists.
EEXIST - 你试图创建已存在的消息队列。

EINVAL - An inappropriate name was given for the message queue, or the values of mq_maxmsg or mq_msgsize were less than 0.
EINVAL - 不适当的名称被赋给消息队列，或 mq_maxmsg 或 mq_msgsize 的值小于 0。

ENOENT - The message queue does not exist, and you did not specify to create it.
ENOENT - 消息队列不存在，并且你未指定要创建它。

EINTR - The call to mq_open was interrupted by a signal.
EINTR - 调用 mq_open 被一个信号中断。

EMFILE - The process has too many files or message queues open. This is a process limit error.
EMFILE - 进程有太多的文件或消息队列已打开。这是进程限制错误。

ENFILE - The system has run out of resources to support more open message queues. This is a system error.
ENFILE - 系统已用完支持更多打开消息队列的资源。这是系统错误。

ENAMETOOLONG - mq_name is too long.
ENAMETOOLONG - mq_name 太长。

DESCRIPTION: 描述：

The mq_open() function establishes the connection between a process and a message queue with a message queue descriptor. If the message queue already exists, mq_open opens it, if the message queue does not exist, mq_open creates it. Message queues can have multiple senders and receivers. If mq_open is successful, the function returns a message queue descriptor. Otherwise, the function returns a -1 and sets ’errno’ to indicate the error.
mq_open()函数建立进程和具有消息队列描述符的消息队列之间的连接。如果消息队列已存在，mq_open 打开它，如果消息队列不存在，mq_open 创建它。消息队列可以有多个发送器和接收器。如果 mq_open 成功，该函数返回消息队列描述符。否则，该函数返回-1 并设置 errno 以表示错误。

The name of the message queue is used as an argument. For the best of portability, the name of the message queue should begin with a "/" and no other "/" should be in the name. Different systems interpret the name in different ways.
消息队列的名称作为参数使用。为了最好的可移植性，消息队列的名称应以“/”开始，并且在名称中没有其他“/”。不同的系统以不同的方式解释名称。

The oflags contain information on how the message is opened if the queue already exists. This may be O_RDONLY for read only, O_WRONLY for write only, of O_RDWR, for read and write.
如果队列已存在，oflags 包含如何打开消息的信息。对于只读，这是 O_RDONLY，对于只写，这是 O_WRONLY，对于读和写，这是 O_RDWR。

In addition, the oflags contain information needed in the creation of a message queue. O_NONBLOCK - If the non-block flag is set then the message queue is non-blocking, and requests to send and receive messages do not block waiting for resources. If the flag is not set then the message queue is blocking, and a request to send might have to wait for an empty message queue. Similarly, a request to receive might have to wait for a message to arrive on the queue. O_CREAT - This call specifies that the call the mq_open is to create a new message queue. In this case the mode

and attribute arguments of the function call are utilized. The message queue is created with a mode similar to the creation of a file, read and write permission creator, group, and others.
此外，oflags 包含在创建消息队列时所需的信息。O_NONBLOCK - 如果设置了非阻塞标志，则消息队列是非阻塞的，发送和接收消息的请求不阻塞等待的资源。如果未设置该标志，则消息队列被阻塞，发送的请求可能必须等待空消息队列。类似地，接收的请求可能必须等待消息到达队列。O_CREAT - 此调用指定调用 mq_open 创建新消息队列。在这种情况下，利用函数调用的模式和属性参数。使用类似于创建文件、读取和写入权限、创建者、组等等的模式创建消息队列。

The geometry of the message queue is contained in the attribute structure. This includes mq_msgsize that dictates the maximum size of a single message, and the mq_maxmsg that dictates the maximum number of messages the queue can hold at one time. If a NULL is used in the mq_attr argument, then the message queue is created with implementation defined defaults. O_EXCL - is always set if O_CREAT flag is set. If the message queue already exists, O_EXCL causes an error message to be returned, otherwise, the new message queue fails and appends to the existing one.
消息队列的几何被包含在属性结构中。这包含表示单个消息的最大大小的 mq_msgsize，和表示队列可以同时持有的消息的最大数量的 mq_maxmsg。如果在 mq_attr 参数中使用 NULL，则使用默认定义的实现创建消息队列。O_EXCL - 如果设置了 O_CREAT，总是设置它。如果消息队列已存在，O_EXCL 导致错误消息被返回，否则，新消息队列失败，并添加到现有的一个。

NOTES: 注意事项：

The mq_open() function does not add or remove messages from the queue. When a new message queue is being created, the mq_flag field of the attribute structure is not used.
mq_open()函数不会从队列添加或移除消息。当创建新消息队列时，不使用属性结构的 mq_flag 字段。

16.4.2 mq_close - Close a Message Queue

16.4.2 mq_close - 关闭消息队列

CALLING SEQUENCE: 调用序列：

```
#include <mqueue.h>

int mq_close(
   mqd_t mqdes
);
```

STATUS CODES: 状态码：

EINVAL - The descriptor does not represent a valid open message queue
EINVAL - 描述符没有代表有效的打开消息队列。

DESCRIPTION: 描述：

The mq_close function removes the association between the message queue descriptor, mqdes, and its message queue. If mq_close() is successfully completed, the function returns a value of zero; otherwise, the function returns a value of -1 and sets errno to indicate the error.
mq_close 函数移除消息队列描述符、mqdes 及其消息队列之间的关联。如果成功完成了 mq_close()，该函数返回 0 值；否则，该函数返回-1 并设置 errno 以表示错误。

NOTES: 注意事项：

If the process had successfully attached a notification request to the message queue via mq_notify, this attachment is removed, and the message queue is available for another process to attach for notification. mq_close has no effect on the contents of the message queue, all the messages that were in the queue remain in the queue.
如果进程已通过 mq_notify 成功连接到消息队列的通知请求，此连接被移除，该消息队列可用于另一个连接通知的进程。mq_close 不影响消息队列的内容，队列中的所有消息都保留在队列中。

16.4.3 mq_unlink - Remove a Message Queue

16.4.3 mq_unlink - 移除消息队列

CALLING SEQUENCE: 调用序列：

```
#include <mqueue.h>

int mq_unlink(
   const char *name
);
```

STATUS CODES: 状态码：

EINVAL - The descriptor does not represent a valid message queue
EINVAL - 描述符没有代表有效的消息队列。

DESCRIPTION: 描述：

The mq_unlink() function removes the named message queue. If the message queue is not open when mq_unlink is called, then the queue is immediately eliminated. Any messages that were on the queue are lost, and the queue can not be opened again. If processes have the queue open when mq_unlink is called, the removal of the queue is delayed until the last process using the queue has finished. However, the name of the message queue is removed so that no other process can open it. Upon successful completion, the function returns a value of zero. Otherwise, the named message queue is not changed by this function call, and the function returns a value of -1 and sets errno to indicate the error.
mq_unlink()函数移除指定名称的消息队列。如果当调用 mq_unlink 时消息队列未打开，则该队列被立即排除。在队列上的任何消息都将丢失，并且不能再次打开该队列。如果当调用 mq_unlink 时进程有打开的队列，队列的移除将延迟，直到使用该队列的最后一个进程已完成。然而，该消息队列的名称被移除，所以没有其他进程可以打开它。成功完成之后，该函数返回 0 值。否则，此函数调用不改变指定名称的消息队列，该函数返回-1 值并设置 errno 以表示错误。

NOTES: 注意事项：

Calls to mq_open() to re-create the message queue may fail until the message queue is actually removed. However, the mq_unlink() call need not block until all references have been closed; it may return immediately.

在实际移除消息队列之前调用 mq_open()以重新创建消息队列可能失败。然而，直到所有引用已关闭，mq_unlink()调用才需要阻塞；它可能立即返回。

16.4.4 mq_send - Send a Message to a Message Queue

16.4.4 mq_send - 发送消息到消息队列

CALLING SEQUENCE: 调用序列：

```
#include<mqueue.h>
int mq_send(
   mqd_t        mqdes,
   const char    *msg_ptr,
   size_t        msg_len,
   unsigned int msg_prio
);
```

STATUS CODES: 状态码：

EBADF - The descriptor does not represent a valid message queue, or the queue was opened for read only O_RDONLY
EBADF - 描述符没有代表有效的消息队列，或以只读方式 O_RDONLY 打开队列。

EINVAL - The value of msg_prio was greater than the MQ_PRIO_MAX.
EINVAL - msg_prio 的值大于 MQ_PRIO_MAX。

EMSGSIZE - The msg_len is greater than the mq_msgsize attribute of the message queue
EMSGSIZE - msg_len 大于消息队列的 mq_msgsize 属性。

EAGAIN - The message queue is non-blocking, and there is no room on the queue for another message as specified by the mq_maxmsg.
EAGAIN - 消息队列是非阻塞的，并且队列上没有空间给 mq_maxmsg 所指定的另一个消息。

EINTR - The message queue is blocking. While the process was waiting for free space on the queue, a signal arrived that interrupted the wait.
EINTR - 消息队列是阻塞的。当进程正在等待队列上的空闲空间时，中断等待的信号到达了。

DESCRIPTION: 描述：

The mq_send() function adds the message pointed to by the argument msg_ptr to

the message queue specified by mqdes. Each message is assigned a priority, from 0 to MQ_PRIO_MAX. MQ_PRIO_MAX is defined in <limits.h> and must be at least 32. Messages are added to the queue in order of their priority. The highest priority message is at the front of the queue.
mq_send()函数添加由参数 msg_ptr 指向的消息到由 mqdes 指定的消息队列。每个消息都被分配优先级，从 0 到 MQ_PRIO_MAX。MQ_PRIO_MAX 是在<limits.h>中定义的，必须至少为 32。消息以它们的优先级顺序被添加到队列。最高优先级的消息处于队列的前面。

The maximum number of messages that a message queue may accept is specified at creation by the mq_maxmsg field of the attribute structure. If this amount is exceeded, the behavior of the process is determined according to what oflag was used when the message queue was opened. If the queue was opened with O_NONBLOCK flag set, then the EAGAIN error is returned. If the O_NONBLOCK flag was not set, the process blocks and waits for space on the queue, unless it is interrupted by a signal.
消息队列可以接受的消息的最大数量是在创建的属性结构的 mq_maxmsg 字段上指定的。如果超过了此数量，进程的行为根据在打开消息队列时使用的什么样的 oflag 确定的。如果使用 O_NONBLOCK 标志设置打开队列，则返回 EAGAIN 错误。如果未设置 O_NONBLOCK 标志，进程阻塞，并等待队列上的空间，除非它被一个信号中断。

Upon successful completion, the mq_send() function returns a value of zero. Otherwise, no message is enqueued, the function returns -1, and errno is set to indicate the error.
成功完成之后，mq_send()函数返回 0 值。否则，没有消息被排除，该函数返回-1，并设置 errno 以表示错误。

NOTES: 注意事项：

If the specified message queue is not full, mq_send inserts the message at the position indicated by the msg_prio argument.
如果指定的消息队列未满，在 msg_prio 参数所指示的位置插入该消息。

16.4.5 mq_receive - Receive a Message from a Message Queue

16.4.5 mq_receive - 从消息队列接收消息

CALLING SEQUENCE: 调用序列：

```
#include <mqueue.h>

size_t   mq_receive(
   mqd_t   mqdes,
   char    *msg_ptr,
   size_t   msg_len,
   unsigned int *msg_prio
);
```

STATUS CODES: 状态码：

EBADF - The descriptor does not represent a valid message queue, or the queue was opened for write only O_WRONLY
EBADF - 描述符没有代表有效的消息队列，或以只读方式 O_WRONLY 打开队列。

EMSGSIZE - The msg_len is less than the mq_msgsize attribute of the message queue
EMSGSIZE - msg_len 小于消息队列的 mq_msgsize 属性。

EAGAIN - The message queue is non-blocking, and the queue is empty
EAGAIN - 消息队列是非阻塞的，并且队列是空的。

EINTR - The message queue is blocking. While the process was waiting for a message to arrive on the queue, a signal arrived that interrupted the wait.
EINTR - 消息队列是阻塞的。当进程正在等待消息到达队列时，中断等待的信号到达了。

DESCRIPTION: 描述：

The mq_receive function is used to receive the oldest of the highest priority message(s) from the message queue specified by mqdes. The messages are received in FIFO order within the priorities. The received message’s priority is stored in the location referenced by the msg_prio. If the msg_prio is a NULL, the priority is discarded. The message is removed and stored in an area pointed to by msg_ptr whose length is of msg_len. The msg_len must be at least equal to the mq_msgsize

attribute of the message queue.
mq_receive 函数用于从 mqdes 所指定的消息队列接收最旧的最高优先级的消息。在优先级内以 FIFO 顺序接收消息。在 msg_prio 所引用的位置中存储接收到的消息的优先级。如果 msg_prio 为 NULL，则丢弃该优先级。该消息被移除，并被存储在长度为 msg_len 的由 msg_ptr 指向的区域中。msg_len 必须至少等于消息队列的 mq_msgsize 属性。

The blocking behavior of the message queue is set by O_NONBLOCK at mq_open or by setting O_NONBLOCK in mq_flags in a call to mq_setattr. If this is a blocking queue, the process blocks and waits on an empty queue. If this a non-blocking queue, the process does not block.
消息队列的阻塞行为是由 mq_open 中的 O_NONBLOCK 或由调用 mq_setattr 时的 mq_flags 中的设置 O_NONBLOCK 设置的。如果这是阻塞队列，进程阻塞并等待在空队列。如果这是非阻塞队列，进程不阻塞。

Upon successful completion, mq_receive returns the length of the selected message in bytes and the message is removed from the queue. Otherwise, no message is removed from the queue, the function returns a value of -1, and sets errno to indicate the error.
成功完成之后，mq_receive 返回以字节为单位的所选消息的长度，并从队列移除该消息。否则，没有消息从队列移除，该函数返回-1 值，并设置 errno 以表示错误。

NOTES: 注意事项：

If the size of the buffer in bytes, specified by the msg_len argument, is less than the mq_msgsize attribute of the message queue, the function fails and returns an error
如果由 msg_len 参数指定的以字节为单位的缓冲区大小小于消息队列的 mq_msgsize 属性，该函数失败并返回错误。

16.4.6 mq_notify - Notify Process that a Message is Available

16.4.6 mq_notify - 通知进程消息可用

CALLING SEQUENCE: 调用序列：

```
#include <mqueue.h>

int mq_notify(
   mqd_t   mqdes,
   const struct sigevent *notification
);
```

STATUS CODES: 状态码：

EBADF - The descriptor does not refer to a valid message queue
EBADF - 描述符没有引用到有效的消息队列。

EBUSY - A notification request is already attached to the queue
EBUSY - 通知请求已被连接到队列。

DESCRIPTION: 描述：

If the argument notification is not NULL, this function registers the calling process to be notified of message arrival at an empty message queue associated with the specified message queue descriptor, mqdes.
如果参数 notification 不是 NULL，此函数注册调用进程，以通知消息到达了与指定的消息队列描述符 mqdes 相关联的空消息队列。

Every message queue has the ability to notify one (and only one) process whenever the queue's state changes from empty (0 messages) to nonempty. This means that the process does not have to block or constantly poll while it waits for a message. By calling mq_notify, a notification request is attached to a message queue. When a message is received by an empty queue, if there are no processes blocked and waiting for the message, then the queue notifies the requesting process of a message arrival. There is only one signal sent by the message queue, after that the notification request is de-registered and another process can attach its notification request. After receipt of a notification, a process must re-register if it wishes to be notified again.
每当队列状态从空（0 消息）变为非空时，每个消息队列都有通知一个（且只有

一个）进程的能力。这意味着进程不必在等待消息时阻塞或不断地轮询。通过调用 mq_notify，你可以连接通知请求到消息队列，当消息由空队列接收时，如果没有进程阻塞并等待消息，则队列通知消息到达的请求进程。只有一个信号通过消息队列发送，然后通知请求被取消注册，另一个进程可以连接其通知请求。在收到通知之后，如果进程希望再次被通知，它必须重新注册。

If there is a process blocked and waiting for the message, that process gets the message, and notification is not be sent. Only one process can have a notification request attached to a message queue at any one time. If another process attempts to register a notification request, it fails. You can de-register for a message queue by passing a NULL to mq_notify; this removes any notification request attached to the queue. Whenever the message queue is closed, all notification attachments are removed.
如果有进程阻塞并等待消息，该进程获得消息，并且不发送通知。在任何一个时间，只有一个进程可以有通知请求连接到消息队列。如果另一个进程试图注册通知请求，它将失败。你可以通过传递 NULL 到 mq_notify，取消注册消息队列，这将移除连接到该队列的所有通知请求。每当关闭消息队列时，移除所有通知连接。

Upon successful completion, mq_notify returns a value of zero; otherwise, the function returns a value of -1 and sets errno to indicate the error.
成功完成之后，mq_notify 返回 0 值；否则，该函数返回-1 值并设置 errno 以表示错误。

NOTES:　注意事项：

It is possible for another process to receive the message after the notification is sent but before the notified process has sent its receive request.
可能的是，在发送通知之后，但在通知的进程已发送其接收请求之前，另一个进程接收了该消息。

16.4.7 mq_setattr - Set Message Queue Attributes

16.4.7 mq_setattr - 设置消息队列属性

CALLING SEQUENCE: 调用序列：

```
#include <mqueue.h>

int mq_setattr(
   mqd_t    mqdes,
   const    struct    mq_attr    *mqstat,
   struct mq_attr    *omqstat
);
```

STATUS CODES: 状态码：

EBADF - The message queue descriptor does not refer to a valid, open queue.
EBADF - 消息队列描述符没有引用到有效的打开队列。

EINVAL - The mq_flag value is invalid.
EINVAL - mq_flag 值无效。

DESCRIPTION: 描述：

The mq_setattr function is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by mqdes. The *omqstat represents the old or previous attributes. If omqstat is non-NULL, the function mq_setattr() stores, in the location referenced by omqstat, the previous message queue attributes and the current queue status. These values are the same as would be returned by a call to mq_getattr() at that point.
mq_setattr()函数用于设置与 mqdes 所指定的消息队列描述符所引用的打开消息队列描述相关联的属性。*omqstat 代表早的或先前的属性。如果 omqstat 为非NULL，函数 mq_setattr()存储在 omqstat、先前的消息队列属性和当前队列的状态所引用的位置。这些值同样是通过在此时调用 mq_getattr()返回。

There is only one mq_attr.mq_flag which can be altered by this call. This is the flag that deals with the blocking and non-blocking behavior of the message queue. If the flag is set then the message queue is non-blocking, and requests to send or receive do not block while waiting for resources. If the flag is not set, then message send and receive may involve waiting for an empty queue or waiting for a message to arrive.

只有一个可以由此调用改变的 mq_attr.mq_flag。这是处理消息队列的阻塞和非阻塞行为的标志。如果设置了该标志，则消息队列是非阻塞的，当等待资源时，发送或接收的请求不会阻塞。如果未设置该标志，则消息发送和接收可能涉及等待空队列或等待消息的到达。

Upon successful completion, the function returns a value of zero and the attributes of the message queue have been changed as specified. Otherwise, the message queue attributes is unchanged, and the function returns a value of -1 and sets errno to indicate the error.
成功完成之后，该函数返回 0 值，消息队列属性已被更改为指定的。否则，消息队列属性保持不变，该函数返回-1 值并设置 errno 以表示错误。

NOTES: 注意事项：

All other fields in the mq_attr are ignored by this call.
mq_attr 中的所有其他字段被此调用忽略。

16.4.8 mq_getattr - Get Message Queue Attributes

16.4.8 mq_getattr - 获得消息队列属性

CALLING SEQUENCE: 调用序列：

```
#include <mqueue.h>
int mq_getattr(
   mqd_t mqdes,
   struct mq_attr *mqstat
);
```

STATUS CODES: 状态码：

EBADF - The message queue descriptor does not refer to a valid, open message queue.
EBADF - 消息队列描述符没有引用到有效的打开消息队列。

DESCRIPTION: 描述：

The mqdes argument specifies a message queue descriptor. The mq_getattr function is used to get status information and attributes of the message queue associated with the message queue descriptor. The results are returned in the mq_attr structure referenced by the mqstat argument. All of these attributes are set at create time, except the blocking/non-blocking behavior of the message queue which can be dynamically set by using mq_setattr. The attribute mq_curmsg is set to reflect the number of messages on the queue at the time that mq_getattr was called.
mqdes 参数指定消息队列描述符。mq_getattr()函数用于获得与消息队列描述符相关联的消息队列的状态信息和属性。在 mqstat 参数所引用的 mq_attr 结构中返回结果。所有这些属性在创建时被设置，除了可以通过使用 mq_setattr 动态设置的消息队列的阻塞/非阻塞行为。属性 mq_curmsg 被设置，以反映在调用 mq_getattr 时在队列上的消息的数量。

Upon successful completion, the mq_getattr function returns zero. Otherwise, the function returns -1 and sets errno to indicate the error.
成功完成之后，mq_getattr 函数返回 0。否则，该函数返回-1 并设置 errno 以表示错误。

NOTES: 注意事项：

17 Thread Manager

17 线程管理器

17.1 Introduction

17.1 简介

The thread manager implements the functionality required of the thread manager as defined by POSIX 1003.1b-1996. This standard requires that a compliant operating system provide the facilities to manage multiple threads of control and defines the API that must be provided.
线程管理器实现由 POSIX 1003.1b-1996 定义的线程管理器的所需功能。此标准要求兼容操作系统提供机制，管理控制的多个线程，并定义必须提供的 API。

The services provided by the thread manager are:
由线程管理器提供的服务有：

- pthread_attr_init - Initialize a Thread Attribute Set
- pthread_attr_destroy - Destroy a Thread Attribute Set
- pthread_attr_setdetachstate - Set Detach State
- pthread_attr_getdetachstate - Get Detach State
- pthread_attr_setstacksize - Set Thread Stack Size
- pthread_attr_getstacksize - Get Thread Stack Size
- pthread_attr_setstackaddr - Set Thread Stack Address
- pthread_attr_getstackaddr - Get Thread Stack Address
- pthread_attr_setscope - Set Thread Scheduling Scope
- pthread_attr_getscope - Get Thread Scheduling Scope
- pthread_attr_setinheritsched - Set Inherit Scheduler Flag
- pthread_attr_getinheritsched - Get Inherit Scheduler Flag
- pthread_attr_setschedpolicy - Set Scheduling Policy
- pthread_attr_getschedpolicy - Get Scheduling Policy
- pthread_attr_setschedparam - Set Scheduling Parameters
- pthread_attr_getschedparam - Get Scheduling Parameters
- pthread_create - Create a Thread
- pthread_exit - Terminate the Current Thread
- pthread_detach - Detach a Thread
- pthread_join - Wait for Thread Termination
- pthread_self - Get Thread ID

- pthread_equal - Compare Thread IDs
- pthread_once - Dynamic Package Initialization
- pthread_setschedparam - Set Thread Scheduling Parameters
- pthread_getschedparam - Get Thread Scheduling Parameters

- pthread_attr_init - 初始化线程属性集
- pthread_attr_destroy - 销毁线程属性集
- pthread_attr_setdetachstate - 设置分离状态
- pthread_attr_getdetachstate - 获得分离状态
- pthread_attr_setstacksize - 设置线程栈大小
- pthread_attr_getstacksize - 获得线程栈大小
- pthread_attr_setstackaddr - 设置线程栈地址
- pthread_attr_getstackaddr - 获得线程栈地址
- pthread_attr_setscope - 设置线程调度作用域
- pthread_attr_getscope - 获得线程调度作用域
- pthread_attr_setinheritsched - 设置继承调度器标志
- pthread_attr_getinheritsched - 获得继承调度器标志
- pthread_attr_setschedpolicy - 设置调度策略
- pthread_attr_getschedpolicy - 获得调度策略
- pthread_attr_setschedparam - 设置调度参数
- pthread_attr_getschedparam - 获得调度参数
- pthread_create - 创建线程
- pthread_exit - 终止当前线程
- pthread_detach - 分离线程
- pthread_join - 等待线程终止
- pthread_self - 获得线程 ID
- pthread_equal - 比较线程 ID
- pthread_once - 动态包初始化
- pthread_setschedparam - 设置线程调度参数
- pthread_getschedparam - 获得线程调度参数

17.2 Background

17.2 背景知识

17.2.1 Thread Attributes

17.2.1 线程属性

Thread attributes are utilized only at thread creation time. A thread attribute structure may be initialized and passed as an argument to the pthread_create routine.
线程属性仅在线程创建时被利用。线程属性结构可以被初始化，并作为参数传递给 pthread_create 例程。

stack address	is the address of the optionally user specified stack area for this thread. If this value is NULL, then RTEMS allocates the memory for the thread stack from the RTEMS Workspace Area. Otherwise, this is the user specified address for the memory to be used for the thread's stack. Each thread must have a distinct stack area. Each processor family has different alignment rules which should be followed.
栈地址	是可选的用户指定的此线程的栈区域的地址。如果此值为 NULL，则 RTEMS 从 RTEMS 工作区为线程栈分配内存。否则，这是用户指定的用于该线程栈的内存地址。每个线程必须有不同的栈区域。每个处理器家族有应遵循的不同的对齐规则。
stack size	is the minimum desired size for this thread's stack area. If the size of this area as specified by the stack size attribute is smaller than the minimum for this processor family and the stack is not user specified, then RTEMS will automatically allocate a stack of the minimum size for this processor family.
栈大小	是此线程的栈区域的期望的最小大小。如果由栈大小属性指定的此区域大小小于此处理器家族的最小值，并且栈不是用户指定的，则 RTEMS 将为此处理器家族自动分配最小大小的栈。
contention scope	specifies the scheduling contention scope. RTEMS only supports the PTHREAD_SCOPE_PROCESS scheduling contention scope.
竞争作用域	指定调度竞争作用域。RTEMS 仅支持 PTHREAD_SCOPE_PROCESS 调度竞争作用域。

scheduling inheritance

specifies whether a user specified or the scheduling policy and parameters of the currently executing thread are to be used. When this is PTHREAD_INHERIT_SCHED, then the scheduling policy and parameters of the currently executing thread are inherited by the newly created thread.

调度继承 指定是用户指定，还是调度器策略和当前执行线程的参数要被使用。当这是 PTHREAD_INHERIT_SCHED 时，调度器策略和当前执行线程的参数被新创建的线程继承。

scheduling policy and parameters

specify the manner in which the thread will contend for the processor. The scheduling parameters are interpreted based on the specified policy. All policies utilize the thread priority parameter.

调度策略和参数 指定线程竞争处理器的方式。根据指定的策略解释调度参数。所有策略都利用线程优先级参数。

17.3 Operations

17.3 操作

There is currently no text in this section.
当前在本节中没有文本。

17.4 Services

17.4 服务

This section details the thread manager’s services. A subsection is dedicated to each of this manager’s services and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了线程管理器的服务。一小节专注于该管理器的一个服务，描述了调用序列、相关的常量、用法和状态码。

17.4.1 pthread_attr_init - Initialize a Thread Attribute Set

17.4.1 pthread_attr_init - 初始化线程属性集

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_init(
   pthread_attr_t *attr
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。

DESCRIPTION: 描述：

The pthread_attr_init routine initializes the thread attributes object specified by attr with the default value for all of the individual attributes.
pthread_attr_init 例程使用所有个别属性的默认值初始化由 attr 指定的线程属性对象。

NOTES: 注意事项：

The settings in the default attributes are implementation defined. For RTEMS, the default attributes are as follows:
默认属性中的设置是实现定义的。对于 RTEMS，默认属性如下所示：

- stackadr is not set to indicate that RTEMS is to allocate the stack memory.
- stacksize is set to PTHREAD_MINIMUM_STACK_SIZE.
- contentionscope is set to PTHREAD_SCOPE_PROCESS.
- inheritsched is set to PTHREAD_INHERIT_SCHED to indicate that the created thread inherits its scheduling attributes from its parent.
- detachstate is set to PTHREAD_CREATE_JOINABLE.

- stackadr 未设置，以表示 RTEMS 要分配栈内存。
- stacksize 被设置为 PTHREAD_MINIMUM_STACK_SIZE。
- contentionscope 被设置为 PTHREAD_SCOPE_PROCESS。

- inheritsched 被设置为 PTHREAD_INHERIT_SCHED，以表示创建的线程从其父线程继承调度属性。
- detachstate 被设置为 PTHREAD_CREATE_JOINABLE。

17.4.2 pthread_attr_destroy - Destroy a Thread Attribute Set

17.4.2 pthread_attr_destroy - 销毁线程属性集

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_destroy(
   pthread_attr_t *attr
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。

DESCRIPTION: 描述：

The pthread_attr_destroy routine is used to destroy a thread attributes object. The behavior of using an attributes object after it is destroyed is implementation dependent.
pthread_attr_destroy 例程用于销毁线程属性对象。在属性对象被销毁之后使用属性对象的行为是实现依赖的。

NOTES: 注意事项：

NONE
无

17.4.3 pthread_attr_setdetachstate - Set Detach State

17.4.3 pthread_attr_setdetachstate - 设置分离状态

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int   pthread_attr_setdetachstate(
   pthread_attr_t *attr,
   int   detachstate
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The detachstate argument is invalid.
EINVAL detachstate 参数无效。

DESCRIPTION: 描述：

The pthread_attr_setdetachstate routine is used to value of the detachstate attribute. This attribute controls whether the thread is created in a detached state.
pthread_attr_setdetachstate 例程用于 detachstate 属性的值。此属性控制是否在分离状态中创建线程。

The detachstate can be either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE. The default value for all threads is PTHREAD_CREATE_JOINABLE.
detachstate 可以是 PTHREAD_CREATE_DETACHED 或 PTHREAD_CREATE_JOINABLE。所有线程的默认值是 PTHREAD_CREATE_JOINABLE。

NOTES: 注意事项：

If a thread is in a detached state, then the use of the ID with the pthread_detach or

pthread_join routines is an error.
如果线程处于分离状态中，则使用 pthread_detach 或 pthread_join 例程的 ID 是错误的。

17.4.4 pthread_attr_getdetachstate - Get Detach State

17.4.4 pthread_attr_getdetachstate - 获得分离状态

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_getdetachstate(
   const pthread_attr_t *attr,
   int    *detachstate
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The detatchstate pointer argument is invalid.
EINVAL detatchstate 指针参数无效。

DESCRIPTION: 描述：

The pthread_attr_getdetachstate routine is used to obtain the current value of the detachstate attribute as specified by the attr thread attribute object.
pthread_attr_getdetachstate 例程用于获取由 attr 线程属性对象指定的 detachstate 属性的当前值。

NOTES: 注意事项：

NONE
无

17.4.5 pthread_attr_setstacksize - Set Thread Stack Size

17.4.5 pthread_attr_setstacksize - 设置线程栈大小

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_setstacksize(
   pthread_attr_t *attr,
   size_t    stacksize
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

DESCRIPTION: 描述：

The pthread_attr_setstacksize routine is used to set the stacksize attribute in the attr thread attribute object.
pthread_attr_setstacksize 例程用于设置 attr 线程属性对象中的 stacksize 属性。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_ATTR_STACKSIZE to indicate that this routine is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_ATTR_STACKSIZE，以表示此例程被支持。

If the specified stacksize is below the minimum required for this CPU (PTHREAD_STACK_MIN, then the stacksize will be set to the minimum for this CPU.
如果指定的栈大小低于此 CPU 的最低要求（PTHREAD_STACK_MIN），则栈大小将被设置为此 CPU 的最小值。

17.4.6 pthread_attr_getstacksize - Get Thread Stack Size

17.4.6 pthread_attr_getstacksize - 获得线程栈大小

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_attr_getstacksize(
   const pthread_attr_t *attr,
   size_t    *stacksize
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The stacksize pointer argument is invalid.
EINVAL stacksize 指针参数无效。

DESCRIPTION: 描述：

The pthread_attr_getstacksize routine is used to obtain the stacksize attribute in the attr thread attribute object.
pthread_attr_getstacksize 例程用于获取 attr 线程属性对象中的 stacksize 属性。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_ATTR_STACKSIZE to indicate that this routine is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_ATTR_STACKSIZE，以表示此例程被支持。

17.4.7 pthread_attr_setstackaddr - Set Thread Stack Address

17.4.7 pthread_attr_setstackaddr - 设置线程栈地址

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_setstackaddr(
   pthread_attr_t *attr,
   void    *stackaddr
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。

DESCRIPTION: 描述：

The pthread_attr_setstackaddr routine is used to set the stackaddr attribute in the attr thread attribute object.
pthread_attr_setstackaddr 例程用于设置 attr 线程属性对象中的 stackaddr 属性。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_ATTR_STACKADDR to indicate that this routine is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_ATTR_STACKADDR，以表示此例程被支持。

It is imperative to the proper operation of the system that each thread have sufficient stack space.
对正确的系统操作必要的是，每个线程有足够的栈空间。

17.4.8 pthread_attr_getstackaddr - Get Thread Stack Address

17.4.8 pthread_attr_getstackaddr - 获得线程栈地址

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_getstackaddr(
   const pthread_attr_t       *attr,
   void    **stackaddr
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The stackaddr pointer argument is invalid.
EINVAL stackaddr 指针参数无效。

DESCRIPTION: 描述：

The pthread_attr_getstackaddr routine is used to obtain the stackaddr attribute in the attr thread attribute object.
pthread_attr_getstackaddr 例程用于获取 attr 线程属性对象中的 stackaddr 属性。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_ATTR_STACKADDR to indicate that this routine is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_ATTR_STACKADDR，以表示此例程被支持。

17.4.9 pthread_attr_setscope - Set Thread Scheduling Scope

17.4.9 pthread_attr_setscope - 设置线程调度作用域

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_setscope(
   pthread_attr_t *attr,
   int    contentionscope
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The contention scope specified is not valid.
EINVAL 指定的竞争作用域无效。

ENOTSUP The contention scope specified (PTHREAD_SCOPE_SYSTEM) is not supported.
ENOTSUP 指定的竞争作用域（PTHREAD_SCOPE_SYSTEM）不被支持。

DESCRIPTION: 描述：

The pthread_attr_setscope routine is used to set the contention scope field in the thread attribute object attr to the value specified by contentionscope.
pthread_attr_setscope 例程用于设置线程属性对象 attr 中的竞争作用域字段为 contentionscope 所指定的值。

The contentionscope must be either PTHREAD_SCOPE_SYSTEM to indicate that the thread is to be within system scheduling contention or PTHREAD_SCOPE_PROCESS indicating that the thread is to be within the process scheduling contention scope.
contentionscope 必须是 PTHREAD_SCOPE_SYSTEM 以表示该线程在系统调度竞争内，或是 PTHREAD_SCOPE_PROCESS 以表示该线程在进程调度竞争作用域内。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.

按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.10 pthread_attr_getscope - Get Thread Scheduling Scope

17.4.10 pthread_attr_getscope - 获得线程调度作用域

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_getscope(
   const pthread_attr_t *attr,
   int    *contentionscope
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。
EINVAL	The contentionscope pointer argument is invalid.
EINVAL	contentionscope 指针参数无效。

DESCRIPTION: 描述：

The pthread_attr_getscope routine is used to obtain the value of the contention scope field in the thread attributes object attr. The current value is returned in contentionscope.
pthread_attr_getscope 例程用于获取线程属性对象 attr 中的竞争作用域字段的值。在 contentionscope 返回当前值。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.11 pthread_attr_setinheritsched - Set Inherit Scheduler Flag

17.4.11 pthread_attr_setinheritsched - 设置继承调度器标志

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_setinheritsched(
   pthread_attr_t *attr,
   int    inheritsched
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The specified scheduler inheritance argument is invalid.
EINVAL 指定的调度器继承参数无效。

DESCRIPTION: 描述：

The pthread_attr_setinheritsched routine is used to set the inherit scheduler field in the thread attribute object attr to the value specified by inheritsched.
pthread_attr_setinheritsched 例程用于设置线程属性对象 attr 中的继承调度器字段为 inheritsched 所指定的值。

The contentionscope must be either PTHREAD_INHERIT_SCHED to indicate that the thread is to inherit the scheduling policy and parameters from the creating thread, or PTHREAD_EXPLICIT_SCHED to indicate that the scheduling policy and parameters for this thread are to be set from the corresponding values in the attributes object. If contentionscope is PTHREAD_INHERIT_SCHED, then the scheduling attributes in the attr structure will be ignored at thread creation time.
contentionscope 必须是 PTHREAD_INHERIT_SCHED 以表示该线程从创建线程继承调度策略和参数，或是 PTHREAD_EXPLICIT_SCHED 以表示此线程的调度策略和参数从属性对象中的对应值设置。如果 contentionscope 是 PTHREAD_INHERIT_SCHED，

则 attr 结构中的调度属性将在线程创建时被忽略。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.12 pthread_attr_getinheritsched - Get Inherit Scheduler Flag

17.4.12 pthread_attr_getinheritsched - 获得继承调度器标志

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_attr_getinheritsched(
   const pthread_attr_t *attr,
   int    *inheritsched
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。
EINVAL	The inheritsched pointer argument is invalid.
EINVAL	inheritsched 指针参数无效。

DESCRIPTION: 描述：

The pthread_attr_getinheritsched routine is used to object the current value of the inherit scheduler field in the thread attribute object attr.
pthread_attr_getinheritsched 例程用于获取线程属性对象 attr 中的继承调度器字段的当前值。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.13 pthread_attr_setschedpolicy - Set Scheduling Policy

17.4.13 pthread_attr_setschedpolicy - 设置调度策略

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_setschedpolicy(
   pthread_attr_t *attr,
   int    policy
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

ENOTSUP The specified scheduler policy argument is invalid.
ENOTSUP 指定的调度器策略参数无效。

DESCRIPTION: 描述：

The pthread_attr_setschedpolicy routine is used to set the scheduler policy field in the thread attribute object attr to the value specified by policy.
pthread_attr_setschedpolicy 例程用于设置线程属性对象 attr 中的调度器策略字段为 policy 所指定的值。

Scheduling policies may be one of the following:
调度器策略可以为下列之一：

- SCHED_DEFAULT
- SCHED_FIFO
- SCHED_RR
- SCHED_SPORADIC
- SCHED_OTHER

The precise meaning of each of these is discussed elsewhere in this manual.
每一项的确切含义在本指南的其他地方讨论。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.14 pthread_attr_getschedpolicy - Get Scheduling Policy

17.4.14 pthread_attr_getschedpolicy - 获得调度策略

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_getschedpolicy(
   const pthread_attr_t *attr,
   int    *policy
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The specified scheduler policy argument pointer is invalid.
EINVAL 指定的调度器策略参数指针无效。

DESCRIPTION: 描述：

The pthread_attr_getschedpolicy routine is used to obtain the scheduler policy field from the thread attribute object attr. The value of this field is returned in policy.
pthread_attr_getschedpolicy 例程用于从线程属性对象 attr 获取调度器策略字段。在 policy 中返回此字段的值。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.15 pthread_attr_setschedparam - Set Scheduling Parameters

17.4.15 pthread_attr_setschedparam - 设置调度参数

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_setschedparam(
   pthread_attr_t    *attr,
   const struct sched_param       param
);
```

STATUS CODES: 状态码：

EINVAL The attribute pointer argument is invalid.
EINVAL 属性指针参数无效。

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The specified scheduler parameter argument is invalid.
EINVAL 指定的调度器参数的参数无效。

DESCRIPTION: 描述：

The pthread_attr_setschedparam routine is used to set the scheduler parameters field in the thread attribute object attr to the value specified by param.
pthread_attr_setschedparam 例程用于设置线程属性对象 attr 中的调度器参数字段为 param 所指定的值。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.16 pthread_attr_getschedparam - Get Scheduling Parameters

17.4.16 pthread_attr_getschedparam - 获得调度参数

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_attr_getschedparam(
   const   pthread_attr_t   *attr,
   struct sched_param   *param
);
```

STATUS CODES: 状态码：

EINVAL	The attribute pointer argument is invalid.
EINVAL	属性指针参数无效。
EINVAL	The attribute set is not initialized.
EINVAL	属性集未初始化。
EINVAL	The specified scheduler parameter argument pointer is invalid.
EINVAL	指定的调度器参数的参数指针无效。

DESCRIPTION: 描述：

The pthread_attr_getschedparam routine is used to obtain the scheduler parameters field from the thread attribute object attr. The value of this field is returned in param.
pthread_attr_getschedparam 例程用于从线程属性对象 attr 获取调度器参数字段。在 param 中返回此字段的值。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.
按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.17 pthread_create - Create a Thread

17.4.17 pthread_create - 创建线程

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_create(
   pthread_t    *thread,
   const pthread_attr_t    *attr,
   void    (*start_routine)( void *),
   void    *arg
);
```

STATUS CODES: 状态码：

EINVAL The attribute set is not initialized.
EINVAL 属性集未初始化。

EINVAL The user specified a stack address and the size of the area was not large enough to meet this processor's minimum stack requirements.
EINVAL 用户指定的栈地址和区域大小不够大，不能满足此处理器的最小栈要求。

EINVAL The specified scheduler inheritance policy was invalid.
EINVAL 指定的调度器继承策略无效。

ENOTSUP The specified contention scope was PTHREAD_SCOPE_PROCESS.
ENOTSUP 指定的竞争作用域为 PTHREAD_SCOPE_PROCESS。

EINVAL The specified thread priority was invalid.
EINVAL 指定的线程优先级无效。

EINVAL The specified scheduling policy was invalid.
EINVAL 指定的调度策略无效。

EINVAL The scheduling policy was SCHED_SPORADIC and the specified replenishment period is less than the initial budget.
EINVAL 调度策略为 SCHED_SPORADIC，并且指定的补充周期小于初始预算。

EINVAL The scheduling policy was SCHED_SPORADIC and the specified low priority is invalid.
EINVAL 调度策略为 SCHED_SPORADIC，并且指定的低优先级无效。

EAGAIN The system lacked the necessary resources to create another thread, or the self imposed limit on the total number of threads in a process PTHREAD_THREAD_MAX would be exceeded.
EAGAIN 系统缺乏必需的资源来创建另一个线程，或超出了自身强加的进程中的线程总数 PTHREAD_THREAD_MAX 的限制。

EINVAL Invalid argument passed.
EINVAL 无效的参数传递。

DESCRIPTION: 描述：

The pthread_create routine is used to create a new thread with the attributes specified by attr. If the attr argument is NULL, then the default attribute set will be used. Modification of the contents of attr after this thread is created does not have an impact on this thread.
pthread_create 例程用于使用由 attr 指定的属性创建新线程。如果 attr 参数为 NULL，则将使用默认属性集。在创建此线程之后修改 attr 的内容对此线程没有影响。

The thread begins execution at the address specified by start_routine with arg as its only argument. If start_routine returns, then it is functionally equivalent to the thread executing the pthread_exit service.
线程使用 arg 作为其唯一的参数在 start_routine 所指定的地址上开始执行。如果 start_routine 返回，则它在功能上相当于线程执行 pthread_exit 服务。

Upon successful completion, the ID of the created thread is returned in the thread argument.
成功完成之后，在 thread 参数中返回创建的线程的 ID。

NOTES: 注意事项：

There is no concept of a single main thread in RTEMS as there is in a tradition UNIX system. POSIX requires that the implicit return of the main thread results in the same effects as if there were a call to exit. This does not occur in RTEMS.
在 RTEMS 中没有像在传统 UNIX 系统中所具有的单个主线程的概念。POSIX 需要隐式地返回主线程，导致像调用 exit 一样的效果。这不会发生在 RTEMS 中。

The signal mask of the newly created thread is inherited from its creator and the set of pending signals for this thread is empty.
新创建的线程的信号掩码从其创建者继承，此线程的挂起信号集为空。

17.4.18 pthread_exit - Terminate the Current Thread

17.4.18 pthread_exit - 终止当前线程

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

void pthread_exit(
   void *status
);
```

STATUS CODES: 状态码：

NONE
无

DESCRIPTION: 描述：

The pthread_exit routine is used to terminate the calling thread. The status is made available to any successful join with the terminating thread.
pthread_exit 例程用于终止调用线程。status 可用于与终止线程的任何成功连接。

When a thread returns from its start routine, it results in an implicit call to the pthread_exit routine with the return value of the function serving as the argument to pthread_exit.
当一个线程从其启动例程返回时，它导致隐式地调用具有充当给 pthread_exit 的参数的函数的返回值的 pthread_exit 例程。

NOTES: 注意事项：

Any cancellation cleanup handlers that have been pushed and not yet popped shall be popped in reverse of the order that they were pushed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, destructors for that data will be invoked.
已被压栈但尚未弹出的任何取消清除处理程序应以它们被压入的相反顺序弹出。在所有取消息清除处理程序都被执行之后，如果线程有任何线程特定的数据，该数据的析构函数将被调用。

Thread termination does not release or free any application visible resources

including but not limited to mutexes, file descriptors, allocated memory, etc.. Similarly, exitting a thread does not result in any process-oriented cleanup activity.
线程的终止不释放（release 或 free）任何应用程序的可见资源，包括但不限于互斥体、文件描述符、分配的内存等等。类似地，退出线程不会导致任何面向过程的清除活动。

There is no concept of a single main thread in RTEMS as there is in a tradition UNIX system. POSIX requires that the implicit return of the main thread results in the same effects as if there were a call to exit. This does not occur in RTEMS.
在 RTEMS 中没有像在传统 UNIX 系统中所具有的单个主线程的概念。POSIX 需要隐式地返回主线程，导致像调用 exit 一样的效果。这不会发生在 RTEMS 中。

All access to any automatic variables allocated by the threads is lost when the thread exits. Thus references (i.e. pointers) to local variables of a thread should not be used in a global manner without care. As a specific example, a pointer to a local variable should NOT be used as the return value.
当线程退出时，所有对线程所分配的自动变量的访问都会丢失。因此引用（即指向）线程的本地变量的指针不应在不谨慎的全局方式中使用。作为具体的示例，指向本地变量的指针不应用作返回值。

17.4.19 pthread_detach - Detach a Thread

17.4.19 pthread_detach - 分离线程

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_detach(
   pthread_t thread
);
```

STATUS CODES: 状态码：

ESRCH The thread specified is invalid.
ESRCH 指定的线程无效。

EINVAL The thread specified is not a joinable thread.
EINVAL 指定的线程不是可连接的线程。

DESCRIPTION: 描述：

The pthread_detach routine is used to to indicate that storage for thread can be reclaimed when the thread terminates without another thread joinging with it.
pthread_detach 例程用于表示当线程终止时没有另一个线程与它连接，线程的存储可以被回收。

NOTES: 注意事项：

If any threads have previously joined with the specified thread, then they will remain joined with that thread. Any subsequent calls to pthread_join on the specified thread will fail.
如果任何线程先前已经与指定的线程连接，则它们将回收与该线程的连接。随后对指定的线程上的 pthread_join 的任何调用都将失败。

17.4.20 pthread_join - Wait for Thread Termination

17.4.20 pthread_join - 等待线程终止

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_join(
   pthread_t       thread,
   void    **value_ptr
);
```

STATUS CODES: 状态码：

ESRCH The thread specified is invalid.
ESRCH 指定的线程无效。

EINVAL The thread specified is not a joinable thread.
EINVAL 指定的线程不是可连接的线程。

EDEADLK A deadlock was detected or thread is the calling thread.
EDEADLK 检测到死锁或线程是调用线程。

DESCRIPTION: 描述：

The pthread_join routine suspends execution of the calling thread until thread terminates. If thread has already terminated, then this routine returns immediately. The value returned by thread (i.e. passed to pthread_exit is returned in value_ptr.
pthread_join 例程调用线程的执行，直到线程终止。如果线程已经终止，则此例程立即返回。由线程返回的值（即传递到 pthread_exit）在 value_ptr 中返回。

When this routine returns, then thread has been terminated.
当此例程返回时，线程已被终止。

NOTES: 注意事项：

The results of multiple simultaneous joins on the same thread is undefined.
在同一线程上的多个同时连接的结果是未定义的。

If any threads have previously joined with the specified thread, then they will remain joined with that thread. Any subsequent calls to pthread_join on the specified thread will fail.
如果任何线程先前已经与指定的线程连接，则它们将回收与该线程的连接。随后对指定的线程上的 pthread_join 的任何调用都将失败。

If value_ptr is NULL, then no value is returned.
如果 value_ptr 为 NULL，则没有值被返回。

17.4.21 pthread_self - Get Thread ID

17.4.21 pthread_self - 获得线程 ID

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

pthread_t pthread_self( void );
```

STATUS CODES: 状态码：

The value returned is the ID of the calling thread.
返回的值是调用线程的 ID。

DESCRIPTION: 描述：

This routine returns the ID of the calling thread.
这个例程返回调用线程的 ID。

NOTES: 注意事项：

NONE
无

17.4.22 pthread_equal - Compare Thread IDs

17.4.22 pthread_equal - 比较线程 ID

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_equal(
   pthread_t t1,
   pthread_t t2
);
```

STATUS CODES: 状态码：

zero The thread ids are not equal.
0 线程 id 不相等。

non-zero The thread ids are equal.
非 0 线程 id 相等。

DESCRIPTION: 描述：

The pthread_equal routine is used to compare two thread IDs and determine if they are equal.
pthread_equal 例程用于比较两个线程 ID，确定它们是否相等。

NOTES: 注意事项：

The behavior is undefined if the thread IDs are not valid.
如果线程 ID 无效，行为是未定义的。

17.4.23 pthread_once - Dynamic Package Initialization

17.4.23 pthread_once - 动态包初始化

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

pthread_once_t    once_control    =    PTHREAD_ONCE_INIT;

int        pthread_once(
   thread_once_t    *once_control,
   void    (*init_routine)(void)
);
```

STATUS CODES: 状态码：

NONE
无

DESCRIPTION: 描述：

The pthread_once routine is used to provide controlled initialization of variables. The first call to pthread_once by any thread with the same once_control will result in the init_routine being invoked with no arguments. Subsequent calls to pthread_once with the same once_control will have no effect.
pthread_once 例程用于提供可控的变量初始化。由具有相同 once_control 的任何线程首次调用 pthread_once 将导致不带参数调用 init_routine。随后调用具有相同 once_control 的 pthread_once 将不起作用。

The init_routine is guaranteed to have run to completion when this routine returns to the caller.
当此例程返回到调用者时，init_routine 保证已完成运行。

NOTES: 注意事项：

The behavior of pthread_once is undefined if once_control is automatic storage (i.e. on a task stack) or is not initialized using PTHREAD_ONCE_INIT.
如果 once_control 是自动存储的(即在任务栈上)或不是使用 PTHREAD_ONCE_INIT 初始化的，pthread_once 的行为是未定义的。

17.4.24 pthread_setschedparam - Set Thread Scheduling Parameters

17.4.24 pthread_setschedparam - 设置线程调度参数

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_setschedparam(
   pthread_t    thread,
   int    policy,
   struct sched_param *param
);
```

STATUS CODES: 状态码：

EINVAL The scheduling parameters indicated by the parameter param is invalid.
EINVAL 由参数 param 表示的调度参数无效。

EINVAL The value specified by policy is invalid.
EINVAL 由 policy 指定的值无效。

EINVAL The scheduling policy was SCHED_SPORADIC and the specified replenishment period is less than the initial budget.
EINVAL 调度器策略是 SCHED_SPORADIC，并且指定的补充周期小于初始预算。

EINVAL The scheduling policy was SCHED_SPORADIC and the specified low priority is invalid.
EINVAL 调度策略是 SCHED_SPORADIC，并且指定的低优先级无效。

ESRCH The thread indicated was invalid.
ESRCH 表示的线程无效。

DESCRIPTION: 描述：

The pthread_setschedparam routine is used to set the scheduler parameters currently associated with the thread specified by thread to the policy specified by policy. The contents of param are interpreted based upon the policy argument.

pthread_setschedparam 例程用于设置当前与 thread 所指定的线程相关联的调度器参数为 policy 所指定的策略。param 的内容根据 policy 参数解释。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.

按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

17.4.25 pthread_getschedparam - Get Thread Scheduling Parameters

17.4.25 pthread_getschedparam - 获得线程调度参数

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

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

STATUS CODES: 状态码：

EINVAL The policy pointer argument is invalid.
EINVAL 策略指针参数无效。

EINVAL The scheduling parameters pointer argument is invalid.
EINVAL 调度参数指针的参数无效。

ESRCH The thread indicated by the parameter thread is invalid.
ESRCH 由参数 thread 表示的线程无效。

DESCRIPTION: 描述：

The pthread_getschedparam routine is used to obtain the scheduler policy and parameters associated with thread. The current policy and associated parameters values returned in policy and param, respectively.
pthread_getschedparam 例程用于获取调度器策略和与 thread 相关联的参数。当前的策略和相关参数值分别在 policy 和 param 中返回。

NOTES: 注意事项：

As required by POSIX, RTEMS defines the feature symbol _POSIX_THREAD_PRIORITY_SCHEDULING to indicate that the family of routines to which this routine belongs is supported.

按照 POSIX 的要求，RTEMS 定义特征符号_POSIX_THREAD_PRIORITY_SCHEDULING，以表示此例程所属的例程家族被支持。

18 Key Manager

18 键管理器

18.1 Introduction

18.1 简介

The key manager ...
键管理器……

The directives provided by the key manager are:
由键管理器提供的指令有：

- pthread_key_create - Create Thread Specific Data Key
- pthread_key_delete - Delete Thread Specific Data Key
- pthread_setspecific - Set Thread Specific Key Value
- pthread_getspecific - Get Thread Specific Key Value

- pthread_key_create - 创建线程特定的数据键
- pthread_key_delete - 删除线程特定的数据键
- pthread_setspecific - 设置线程特定的键值
- pthread_getspecific - 获得线程特定的键值

18.2 Background

18.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

18.3 Operations

18.3 操作

There is currently no text in this section.
当前在本节中没有文本。

18.4 Directives

18.4 指令

This section details the key manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.

本节详细介绍了键管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

18.4.1 pthread_key_create - Create Thread Specific Data Key

18.4.1 pthread_key_create - 创建线程特定的数据键

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int    pthread_key_create(
   pthread_key_t *key,
   void (*destructor)( void )
);
```

STATUS CODES: 状态码：

EAGAIN There were not enough resources available to create another key.
EAGAIN 没有足够的资源可用于创建另一个键。

ENOMEM Insufficient memory exists to create the key.
ENOMEM 现有的内存不足以创建键。

18.4.2 pthread_key_delete - Delete Thread Specific Data Key

18.4.2 pthread_key_delete - 删除线程特定的数据键

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_key_delete(
    pthread_key_t key,
);
```

STATUS CODES: 状态码：

EINVAL	The key was invalid
EINVAL	键无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

18.4.3 pthread_setspecific - Set Thread Specific Key Value

18.4.3 pthread_setspecific - 设置线程特定的键值

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

int pthread_setspecific(
   pthread_key_t key,
   const void *value
);
```

STATUS CODES: 状态码：

EINVAL	The specified key is invalid.
EINVAL	指定的键无效。

DESCRIPTION: 描述：

NOTES: 注意事项：

18.4.4 pthread_getspecific - Get Thread Specific Key Value

18.4.4 pthread_getspecific - 获得线程特定的键值

CALLING SEQUENCE: 调用序列：

```
#include <pthread.h>

void *pthread_getspecific(
   pthread_key_t key
);
```

STATUS CODES: 状态码：

NULL There is no thread-specific data associated with the specified key.
NULL 没有与指定的键相关联的线程特定的数据。

non-NULL The data associated with the specified key.
非 **NULL** 与指定的键相关联的数据。

DESCRIPTION: 描述：

NOTES: 注意事项：

19 Thread Cancellation Manager

19 线程取消管理器

19.1 Introduction

19.1 简介

The thread cancellation manager is ...
线程取消管理器是……

The directives provided by the thread cancellation manager are:
由线程取消管理器提供的指令有：

- pthread_cancel - Cancel Execution of a Thread
- pthread_setcancelstate - Set Cancelability State
- pthread_setcanceltype - Set Cancelability Type
- pthread_testcancel - Create Cancellation Point
- pthread_cleanup_push - Establish Cancellation Handler
- pthread_cleanup_pop - Remove Cancellation Handler

- pthread_cancel - 取消线程的执行
- pthread_setcancelstate - 设置可取消性状态
- pthread_setcanceltype - 设置可取消性类型
- pthread_testcancel - 创建取消点
- pthread_cleanup_push - 建立取消处理程序
- pthread_cleanup_pop - 移除取消处理程序

19.2 Background

19.2 背景知识

There is currently no text in this section.
当前在本节中没有文本。

19.3 Operations

19.3 操作

There is currently no text in this section.

当前在本节中没有文本。

19.4 Directives

19.4 指令

This section details the thread cancellation manager’s directives. A subsection is dedicated to each of this manager’s directives and describes the calling sequence, related constants, usage, and status codes.
本节详细介绍了线程取消管理器的指令。一小节专注于该管理器的一个指令，描述了调用序列、相关的常量、用法和状态码。

19.4.1 pthread_cancel - Cancel Execution of a Thread

19.4.1 pthread_cancel -取消线程的执行

CALLING SEQUENCE: 调用序列：

```
int pthread_cancel(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

19.4.2 pthread_setcancelstate - Set Cancelability State

19.4.2 pthread_setcancelstate - 设置可取消性状态

CALLING SEQUENCE: 调用序列：

```
int pthread_setcancelstate(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

19.4.3 pthread_setcanceltype - Set Cancelability Type

19.4.3 pthread_setcanceltype - 设置可取消性类型

CALLING SEQUENCE: 调用序列：

```
int pthread_setcanceltype(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

19.4.4 pthread_testcancel - Create Cancellation Point

19.4.4 pthread_testcancel - 创建取消点

CALLING SEQUENCE: 调用序列：

```
int pthread_testcancel(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

19.4.5 pthread_cleanup_push - Establish Cancellation Handler

19.4.5 pthread_cleanup_push - 建立取消处理程序

CALLING SEQUENCE: 调用序列：

```
int pthread_cleanup_push(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

19.4.6 pthread_cleanup_pop - Remove Cancellation Handler

19.4.6 pthread_cleanup_pop - 移除取消处理程序

CALLING SEQUENCE: 调用序列：

```
int pthread_cleanup_push(
);
```

STATUS CODES: 状态码：

E The

DESCRIPTION: 描述：

NOTES: 注意事项：

20 Services Provided by C Library (libc)

20 由 C 库（libc）提供的服务

20.1 Introduction

20.1 简介

This section lists the routines that provided by the Newlib C Library.
本章列出了由 Newlib C 库提供的例程。

20.2 Standard Utility Functions (stdlib.h)

20.2 标准工具函数（stdlib.h）

- abort - Abnormal termination of a program
- abs - Integer absolute value (magnitude)
- assert - Macro for Debugging Diagnostics
- atexit - Request execution of functions at program exit
- atof - String to double or float
- atoi - String to integer
- bsearch - Binary search
- calloc - Allocate space for arrays
- div - Divide two integers
- ecvtbuf - Double or float to string of digits
- ecvt - Double or float to string of digits (malloc result)
- __env_lock - Lock environment list for getenv and setenv
- gvcvt - Format double or float as string
- exit - End program execution
- getenv - Look up environment variable
- labs - Long integer absolute value (magnitude)
- ldiv - Divide two long integers
- malloc - Allocate memory
- realloc - Reallocate memory
- free - Free previously allocated memory
- mallinfo - Get information about allocated memory
- __malloc_lock - Lock memory pool for malloc and free
- mbstowcs - Minimal multibyte string to wide string converter
- mblen - Minimal multibyte length

- mbtowc - Minimal multibyte to wide character converter
- qsort - Sort an array
- rand - Pseudo-random numbers
- strtod - String to double or float
- strtol - String to long
- strtoul - String to unsigned long
- system - Execute command string
- wcstombs - Minimal wide string to multibyte string converter
- wctomb - Minimal wide character to multibyte converter

- abort - 异常终止程序
- abs - 整数绝对值（梯度）
- assert - 用于调试诊断的宏
- atexit - 在程序退出时请求函数的执行
- atof - 字符串到双精度或浮点
- atoi - 字符串到整数
- bsearch - 二进制搜索
- calloc - 为数组分配空间
- div - 两个整数相除
- ecvtbuf - 双精度或浮点到数字字符串
- ecvt - 双精度或浮点到数字字符串（malloc 的结果）
- __env_lock - 锁定 getenv 和 setenv 的环境列表
- gvcvt - 格式化双精度或浮点为字符串
- exit - 结束程序的执行
- getenv - 查找环境变量
- labs - 长整数绝对值（梯度）
- ldiv - 两个长整数相除
- malloc - 分配内存
- realloc - 重新分配内存
- free - 释放先前分配的内存
- mallinfo - 获得关于已分配内存的信息
- __malloc_lock - 锁定 malloc 和 free 的内存池
- mbstowcs - 最小多字节字符串到宽字符串的转换器
- mblen - 最小多字节的长度
- mbtowc - 最小多字节到宽字符的转换器
- qsort - 排序数组
- rand - 伪随机数
- strtod - 字符串到双精度或浮点

- strtol - 字符串到长整数
- strtoul - 字符串到无符号长整数
- system - 执行命令字符串
- wcstombs - 最小宽字符串到多字节字符串的转换器
- wctomb - 最小宽字符到多字节的转换器

20.3 Character Type Macros and Functions (ctype.h)

20.3 字符类型的宏和函数（ctype.h）

- isalnum - Alphanumeric character predicate
- isalpha - Alphabetic character predicate
- isascii - ASCII character predicate
- iscntrl - Control character predicate
- isdigit - Decimal digit predicate
- islower - Lower-case character predicate
- isprint - Printable character predicates (isprint, isgraph)
- ispunct - Punctuation character predicate
- isspace - Whitespace character predicate
- isupper - Uppercase character predicate
- isxdigit - Hexadecimal digit predicate
- toascii - Force integers to ASCII range
- tolower - Translate characters to lower case
- toupper - Translate characters to upper case

- isalnum - 字母数字字符的断定
- isalpha - 字母字符的断定
- isascii - ASCII 字符的断定
- iscntrl - 控制字符的断定
- isdigit - 十进制数字的断定
- islower - 小写字符的断定
- isprint - 可打印字符的断定（isprint、isgraph）
- ispunct - 标点字符的断定
- isspace - 空白字符的断定
- isupper - 大写字符的断定
- isxdigit - 十六进制数字的断定
- toascii - 强制整数到 ASCII 范围
- tolower - 转换字符到小写
- toupper - 转换字符到大写

20.4 Input and Output (stdio.h)

20.4 输入和输出（stdio.h）

- clearerr - Clear file or stream error indicator
- fclose - Close a file
- feof - Test for end of file
- ferror - Test whether read/write error has occurred
- fflush - Flush buffered file output
- fgetc - Get a character from a file or stream
- fgetpos - Record position in a stream or file
- fgets - Get character string from a file or stream
- fiprintf - Write formatted output to file (integer only)
- fopen - Open a file
- fdopen - Turn an open file into a stream
- fputc - Write a character on a stream or file
- fputs - Write a character string in a file or stream
- fread - Read array elements from a file
- freopen - Open a file using an existing file descriptor
- fseek - Set file position
- fsetpos - Restore position of a stream or file
- ftell - Return position in a stream or file
- fwrite - Write array elements from memory to a file or stream
- getc - Get a character from a file or stream (macro)
- getchar - Get a character from standard input (macro)
- gets - Get character string from standard input (obsolete)
- iprintf - Write formatted output (integer only)
- mktemp - Generate unused file name
- perror - Print an error message on standard error
- putc - Write a character on a stream or file (macro)
- putchar - Write a character on standard output (macro)
- puts - Write a character string on standard output
- remove - Delete a file’s name
- rename - Rename a file
- rewind - Reinitialize a file or stream
- setbuf - Specify full buffering for a file or stream
- setvbuf - Specify buffering for a file or stream
- siprintf - Write formatted output (integer only)

- printf - Write formatted output
- scanf - Scan and format input
- tmpfile - Create a temporary file
- tmpnam - Generate name for a temporary file
- vprintf - Format variable argument list

- clearerr - 清除文件或流的错误指示器
- fclose - 关闭文件
- feof - 测试文件的结束
- ferror - 测试是否发生了读/写错误
- fflush - 刷新缓冲的文件输出
- fgetc - 从文件或流获得字符
- fgetpos - 记录流或文件中的位置
- fgets - 从文件或流获得字符串
- fiprintf - 写格式化输出到文件（仅限整数）
- fopen - 打开文件
- fdopen - 转换打开文件为流
- fputc - 在流或文件上写入字符
- fputs - 在文件或流中写入字符串
- fread - 从文件读取数组元素
- freopen - 使用现有文件描述符打开文件
- fseek - 设置文件位置
- fsetpos - 恢复流或文件的位置
- ftell - 返回流或文件中的位置
- fwrite - 从内存写入数组元素到文件或流
- getc - 从文件或流获得字符（宏）
- getchar - 从标准输入获得字符（宏）
- gets - 从标准输入获得字符串（已过时）
- iprintf - 写入格式化输出（仅限整数）
- mktemp - 生成未使用的文件名称
- perror - 在标准错误上打印错误消息
- putc - 在流或文件上写入字符（宏）
- putchar - 在标准输出上写入字符（宏）
- puts - 在标准输出上写入字符串
- remove - 删除文件名
- rename - 重命名文件
- rewind - 重新初始化文件或流
- setbuf - 指定文件或流的完全缓冲

- setvbuf - 指定文件或流的缓冲
- siprintf - 写入格式化输出（仅限整数）
- printf - 写入格式化输出
- scanf - 扫描或格式化输入
- tmpfile - 创建临时文件
- tmpnam - 生成临时文件的名称
- vprintf - 格式化变量参数列表

20.5 Strings and Memory (string.h)

20.5 字符串和内存（string.h）

- bcmp - Compare two memory areas
- bcopy - Copy memory regions
- bzero - Initialize memory to zero
- index - Search for character in string
- memchr - Find character in memory
- memcmp - Compare two memory areas
- memcpy - Copy memory regions
- memmove - Move possibly overlapping memory
- memset - Set an area of memory
- rindex - Reverse search for character in string
- strcasecmp - Compare strings ignoring case
- strcat - Concatenate strings
- strchr - Search for character in string
- strcmp - Character string compare
- strcoll - Locale specific character string compare
- strcpy - Copy string
- strcspn - Count chars not in string
- strerror - Convert error number to string
- strlen - Character string length
- strlwr - Convert string to lower case
- strncasecmp - Compare strings ignoring case
- strncat - Concatenate strings
- strncmp - Character string compare
- strncpy - Counted copy string
- strpbrk - Find chars in string
- strrchr - Reverse search for character in string
- strspn - Find initial match

- strstr - Find string segment
- strtok - Get next token from a string
- strupr - Convert string to upper case
- strxfrm - Transform string

- bcmp - 比较两个内存区域
- bcopy - 复制内存区域
- bzero - 初始化内存为 0
- index - 搜索字符串中的字符
- memchr - 在内存中查找字符
- memcmp - 比较两个内存区域
- memcpy - 复制内存区域
- memmove - 移动可能重叠的内存
- memset - 设置内存区域
- rindex - 反向搜索字符串中的字符
- strcasecmp - 比较字符串，忽略大小写
- strcat - 连接字符串
- strchr - 搜索字符串中的字符
- strcmp - 字符串比较
- strcoll - 区域设置特定的字符串比较
- strcpy - 复制字符串
- strcspn - 计数不在字符串中的字符
- strerror - 转换错误号到字符串
- strlen - 字符串长度
- strlwr - 转换字符串到小写
- strncasecmp - 比较字符串，忽略大小写
- strncat - 连接字符串
- strncmp - 字符串比较
- strncpy - 计数地复制字符串
- strpbrk - 查找字符串中的字符
- strrchr - 反向搜索字符串中的字符
- strspn - 查找初始匹配
- strstr - 查找字符串部分
- strtok - 从字符串获得下一个令牌
- strupr - 转换字符串到大写
- strxfrm - 转换字符串

20.6 Signal Handling (signal.h)

20.6 信号处理（signal.h）

- raise - Send a signal
- signal - Specify handler subroutine for a signal

- raise - 发送信号
- signal - 指定信号的处理程序子例程

20.7 Time Functions (time.h)

20.7 时间函数（time.h）

- asctime - Format time as string
- clock - Cumulative processor time
- ctime - Convert time to local and format as string
- difftime - Subtract two times
- gmtime - Convert time to UTC (GMT) traditional representation
- localtime - Convert time to local representation
- mktime - Convert time to arithmetic representation
- strftime - Flexible calendar time formatter
- time - Get current calendar time (as single number)

- asctime - 格式化时间为字符串
- clock - 累加的处理器时间
- ctime - 转换时间到本地并格式化为字符串
- difftime - 两个时间相减
- gmtime - 转换时间到 UTC (GMT)的传统表示法
- localtime - 转换时间到本地表示法
- mktime - 转换时间到算术表示法
- strftime - 灵活的日历时间格式化程序
- time - 获得当前的日历时间（作为单个数字）

20.8 Locale (locale.h)

20.8 区域设置（locale.h）

- setlocale - Select or query locale
- setlocale - 选择或查询区域设置

20.9 Reentrant Versions of Functions

20.9 可重入版本的函数

- Equivalent for errno variable:
- 相当于 errno 变量：

  - errno_r - XXX

- Locale functions:
- 区域设置函数：

  - localeconv_r - XXX
  - setlocale_r - XXX

- Equivalents for stdio variables:
- 相当于 stdio 变量

  - stdin_r - XXX
  - stdout_r - XXX
  - stderr_r - XXX

- Stdio functions:
- Stdio 函数：

  - fdopen_r - XXX
  - perror_r - XXX
  - tempnam_r - XXX
  - fopen_r - XXX
  - putchar_r - XXX
  - tmpnam_r - XXX
  - getchar_r - XXX
  - puts_r - XXX
  - tmpfile_r - XXX
  - gets_r - XXX
  - remove_r - XXX
  - vfprintf_r - XXX
  - iprintf_r - XXX

- rename_r - XXX
- vsnprintf_r - XXX
- mkstemp_r - XXX
- snprintf_r - XXX
- vsprintf_r - XXX
- mktemp_t - XXX
- sprintf_r - XXX

- Signal functions:
- 信号函数：

  - init_signal_r - XXX
  - signal_r - XXX
  - kill_r - XXX
  - _sigtramp_r - XXX
  - raise_r - XXX

- Stdlib functions:
- Stdlib 函数：

  - calloc_r - XXX
  - mblen_r - XXX
  - srand_r - XXX
  - dtoa_r - XXX
  - mbstowcs_r - XXX
  - strtod_r - XXX
  - free_r - XXX
  - mbtowc_r - XXX
  - strtol_r - XXX
  - getenv_r - XXX
  - memalign_r - XXX
  - strtoul_r - XXX
  - mallinfo_r - XXX
  - mstats_r - XXX
  - system_r - XXX
  - malloc_r - XXX
  - rand_r - XXX
  - wcstombs_r - XXX

- malloc_r - XXX
- realloc_r - XXX
- wctomb_r - XXX
- malloc_stats_r - XXX
- setenv_r - XXX

- String functions:
- 字符串函数：

  - strtok_r - XXX

- System functions:
- 系统函数：

  - close_r - XXX
  - link_r - XXX
  - unlink_r - XXX
  - execve_r - XXX
  - lseek_r - XXX
  - wait_r - XXX
  - fcntl_r - XXX
  - open_r - XXX
  - write_r - XXX
  - fork_r - XXX
  - read_r - XXX
  - fstat_r - XXX
  - sbrk_r - XXX
  - gettimeofday_r - XXX
  - stat_r - XXX
  - getpid_r - XXX
  - times_r - XXX

- Time function:
- 时间函数：

  - asctime_r - XXX

20.10 Miscellaneous Macros and Functions

20.10 杂项宏和函数

- unctrl - Return printable representation of a character
- unctrl - 返回字符的可打印表示法

20.11 Variable Argument Lists

20.11 可变参数列表

- Stdarg (stdarg.h):
  - va_start - XXX
  - va_arg - XXX
  - va_end - XXX
- Vararg (varargs.h):
  - va_alist - XXX
  - va_start-trad - XXX
  - va_arg-trad - XXX
  - va_end-trad - XXX

20.12 Reentrant System Calls

20.12 可重入的系统调用

- open_r - XXX
- close_r - XXX
- lseek_r - XXX
- read_r - XXX
- write_r - XXX
- fork_r - XXX
- wait_r - XXX
- stat_r - XXX
- fstat_r - XXX
- link_r - XXX
- unlink_r - XXX
- sbrk_r - XXX

21 Services Provided by the Math Library (libm)

21 由数学库（libm）提供的服务

21.1 Introduction

21.1 简介

This section lists the routines that provided by the Newlib Math Library (libm).
本章列出了由 Newlib 数学库（libm）提供的例程。

21.2 Standard Math Functions (math.h)

21.2 标准数学函数（math.h）

- acos - Arccosine
- acosh - Inverse hyperbolic cosine
- asin - Arcsine
- asinh - Inverse hyperbolic sine
- atan - Arctangent
- atan2 - Arctangent of y/x
- atanh - Inverse hyperbolic tangent
- jN - Bessel functions (jN and yN)
- cbrt - Cube root
- copysign - Sign of Y and magnitude of X
- cosh - Hyperbolic cosine
- erf - Error function (erf and erfc)
- exp - Exponential
- expm1 - Exponential of x and - 1
- fabs - Absolute value (magnitude)
- floor - Floor and ceiling (floor and ceil)
- fmod - Floating-point remainder (modulo)
- frexp - Split floating-point number
- gamma - Logarithmic gamma function
- hypot - Distance from origin
- ilogb - Get exponent
- infinity - Floating infinity
- isnan - Check type of number
- ldexp - Load exponent

- log - Natural logarithms
- log10 - Base 10 logarithms
- log1p - Log of 1 + X
- matherr - Modifiable math error handler
- modf - Split fractional and integer parts
- nan - Floating Not a Number
- nextafter - Get next representable number
- pow - X to the power Y
- remainder - remainder of X divided by Y
- scalbn - scalbn
- sin - Sine or cosine (sin and cos)
- sinh - Hyperbolic sine
- sqrt - Positive square root
- tan - Tangent
- tanh - Hyperbolic tangent

- acos - 反余弦
- acosh - 反双曲余弦
- asin - 反正弦
- asinh - 反双曲正弦
- atan - 反正切
- atan2 - y/x 的反正切
- atanh - 反双曲正切
- jN - 贝塞尔函数（jN 和 yN）
- cbrt - 立方根
- copysign - Y 符号和 X 梯度
- cosh - 双曲余弦
- erf - 误差函数（erf 和 erfc）
- exp - 指数
- expm1 - x 的指数-1
- fabs - 绝对值（梯度）
- floor - 下限和上限（floor 和 ceil）
- fmod - 浮点余数（除法）
- frexp - 分割浮点数
- gamma - 对数伽玛函数
- hypot - 从原点开始的距离
- ilogb - 获得指数
- infinity - 浮点无穷大

- isnan - 检查数字类型
- ldexp - 加载指数
- log - 自然对数
- log10 - 以 10 为底的对数
- log1p - （1 + X）的对数
- matherr - 可修改的数学错误处理程序
- modf - 分割小数和整数部分
- nan - 浮点不是数字
- nextafter - 获得下一个可表示的数字
- pow - X 的 Y 次幂
- remainder - X 除以 Y 的余数
- scalbn - scalbn
- sin - 正弦或余弦（sin 和 cos）
- sinh - 双曲正弦
- sqrt - 正平方根
- tan - 正切
- tanh - 双曲正切

22 Status of Implementation

22 实现状态

This chapter provides an overview of the status of the implementation of the POSIX API for RTEMS. The POSIX 1003.1b Compliance Guide provides more detailed information regarding the implementation of each of the numerous functions, constants, and macros specified by the POSIX 1003.1b standard.
本章提供了 RTEMS 的 POSIX API 的实现状态的概述。POSIX 1003.1b Compliance Guide 提供了有关 POSIX 1003.1b 标准所指定的许多函数、常量和宏的每个实现的更多详细信息。

RTEMS supports many of the process and user/group oriented services in a "single user/single process" manner. This means that although these services may be of limited usefulness or functionality, they are provided and do work in a coherent manner. This is significant when porting existing code from UNIX to RTEMS.
RTEMS 以“单用户/单进程”的方式支持许多面向服务的进程和用户/组。这意味着虽然这些服务可能是有限的服务或功能，但它们被提供，并以一致的方式工作。当从 UNIX 移植现有代码到 RTEMS 时，这是有意义的。

- Implementation
  - The current implementation of dup() is insufficient.
  - FIFOs mkfifo() are not currently implemented.
  - Asynchronous IO is not implemented.
  - The flockfile() family is not implemented
  - getc/putc unlocked family is not implemented
  - Shared Memory is not implemented
  - Mapped Memory is not implemented
  - NOTES:
    - For Shared Memory and Mapped Memory services, it is unclear what level of support is appropriate and possible for RTEMS.
- Functional Testing
  - Tests for unimplemented services
- Performance Testing
  - There are no POSIX Performance Tests.
- Documentation
  - Many of the service description pages are not complete in this manual. These need to be completed and information added to the background and

operations sections.

- ■ Example programs (not just tests) would be very nice.

- ● 实现
  - ■ dup()的当前实现不足。
  - ■ FIFO mkfifo()当前未实现。
  - ■ 异步 IO 未实现。
  - ■ flockfile()家族未实现。
  - ■ getc/putc 未锁定的家族未实现。
  - ■ 共享内存未实现。
  - ■ 映射内存未实现。
  - ■ 注意：
    - ◆ 对于共享内存和映射内存服务，不清楚什么级别的支持对 RTEMS 来说是适当的和可能的。
- ● 功能测试
  - ■ 测试未实现的服务
- ● 性能测试
  - ■ 没有 POSIX 性能测试。
- ● 文档
  - ■ 本指南中的许多服务描述页是不完整的。这需要完善，信息添加到背景知识和操作部分。
  - ■ 示例程序（不只是测试）将会很好。

Command and Variable Index

命令和变量索引

（未译）

Concept Index

概念索引

（未译）