RTEMS POSIX API User's Guide

Edition 4.0.0, for RTEMS 4.0.0

October 1998

On-Line Applications Research Corporation

On-Line Applications Research Corporation
TEXinfo 1.1.1.1

Any inquiries concerning RTEMS, its related support components, or its documentation should be directed to either:

```
On-Line Applications Research Corporation
4910-L Corporate Drive
Huntsville, AL 35805
VOICE: (256) 722-9985
FAX:   (256) 722-0985
EMAIL: rtems@OARcorp.com
```

Preface

This is the user's guide for the POSIX API support for RTEMS.

We intend for this manual to only be a listing of the API. Please refer to the ISO/IEC 9945-1 *Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]* for the specific functionality and behavior of the services.

In addition, much of the POSIX API standard is actually implemented in the newlib ANSI C Library. Please refer to documentation on newlib for more information on what it supplies.

1 Thread Manager

1.1 Introduction

The thread manager ...

The directives provided by the thread manager are:

- `pthread_attr_init` -
- `pthread_attr_destroy` -
- `pthread_attr_setdetachstate` -
- `pthread_attr_getdetachstate` -
- `pthread_attr_setstacksize` -
- `pthread_attr_getstacksize` -
- `pthread_attr_setstackaddr` -
- `pthread_attr_getstackaddr` -
- `pthread_attr_setscope` -
- `pthread_attr_getscope` -
- `pthread_attr_setinheritsched` -
- `pthread_attr_getinheritsched` -
- `pthread_attr_setschedpolicy` -
- `pthread_attr_getschedpolicy` -
- `pthread_attr_setschedparam` -
- `pthread_attr_getschedparam` -
- `pthread_create` -
- `pthread_exit` -
- `pthread_detach` -
- `pthread_join` -
- `pthread_self` -
- `pthread_equal` -
- `pthread_once` -
- `pthread_setschedparam` -
- `pthread_getschedparam` -

1.2 Background

1.2.1 Thread Attributes

Thread attributes are utilized only at thread creation time.

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.

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.

contention scope specifies the scheduling contention scope. RTEMS only supports the PTHREAD_SCOPE_PROCESS scheduling contention scope.

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.

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.

1.3 Operations

1.4 Directives

This section details the thread 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.

1.4.1 pthread_attr_init

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_init(
  pthread_attr_t  *attr
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.

DESCRIPTION:

NOTES:

1.4.2 pthread_attr_destroy

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_destroy(
  pthread_attr_t  *attr
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.

DESCRIPTION:

NOTES:

1.4.3 pthread_attr_setdetachstate

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_setdetachstate(
  pthread_attr_t  *attr,
  int              detachstate
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The detachstate argument is invalid.

DESCRIPTION:

NOTES:

1.4.4 pthread_attr_getdetachstate

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_getdetachstate(
  const pthread_attr_t  *attr,
  int                   *detachstate
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The detatchstate pointer argument is invalid.

DESCRIPTION:

NOTES:

1.4.5 pthread_attr_setstacksize

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_setstacksize(
  pthread_attr_t  *attr,
  size_t           stacksize
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.

DESCRIPTION:

NOTES:

If the specified stacksize is below the minimum required for this CPU, then the stacksize will be set to the minimum for this CPU.

1.4.6 pthread_attr_getstacksize

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_getstacksize(
  const pthread_attr_t  *attr,
  size_t                *stacksize
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The stacksize pointer argument is invalid.

DESCRIPTION:

NOTES:

1.4.7 pthread_attr_setstackaddr

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_setstackaddr(
  pthread_attr_t  *attr,
  void            *stackaddr
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.

DESCRIPTION:

NOTES:

1.4.8 pthread_attr_getstackaddr

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_getstackaddr(
  const pthread_attr_t  *attr,
  void                 **stackaddr
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The stackaddr pointer argument is invalid.

DESCRIPTION:

NOTES:

1.4.9 pthread_attr_setscope

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_setscope(
  pthread_attr_t  *attr,
  int              contentionscope
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The contention scope specified is not valid.
ENOTSUP	The contention scope specified (PTHREAD_SCOPE_SYSTEM) is not supported.

DESCRIPTION:

NOTES:

1.4.10 pthread_attr_getscope

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_getscope(
  const pthread_attr_t  *attr,
  int                   *contentionscope
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The contentionscope pointer argument is invalid.

DESCRIPTION:

NOTES:

1.4.11 pthread_attr_setinheritsched

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_setinheritsched(
  pthread_attr_t  *attr,
  int              inheritsched
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The specified scheduler inheritance argument is invalid.

DESCRIPTION:

NOTES:

1.4.12 pthread_attr_getinheritsched

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_getinheritsched(
  const pthread_attr_t  *attr,
  int                   *inheritsched
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The inheritsched pointer argument is invalid.

DESCRIPTION:

NOTES:

1.4.13 pthread_attr_setschedpolicy

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_setschedpolicy(
  pthread_attr_t  *attr,
  int              policy
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
ENOTSUP	The specified scheduler policy argument is invalid.

DESCRIPTION:

NOTES:

1.4.14 pthread_attr_getschedpolicy

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_getschedpolicy(
  const pthread_attr_t  *attr,
  int                   *policy
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The specified scheduler policy argument pointer is invalid.

DESCRIPTION:

NOTES:

1.4.15 pthread_attr_setschedparam

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_setschedparam(
  pthread_attr_t            *attr,
  const struct sched_param   param
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The specified scheduler parameter argument is invalid.

DESCRIPTION:

NOTES:

1.4.16 pthread_attr_getschedparam

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_attr_getschedparam(
  const pthread_attr_t  *attr,
  struct sched_param    *param
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The specified scheduler parameter argument pointer is invalid.

DESCRIPTION:

NOTES:

1.4.17 pthread_create

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_create(
  pthread_t            *thread,
  const pthread_attr_t  *attr,
  void                (*start_routine)( void * ),
  void                 *arg
);
```

STATUS CODES:

EINVAL	The attribute set is not initialized.
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	The specified scheduler inheritance policy was invalid.
ENOTSUP	The specified contention scope was PTHREAD_SCOPE_PROCESS.
EINVAL	The specified thread priority was invalid.
EINVAL	The specified scheduling policy was invalid.
EINVAL	The scheduling policy was SCHED_SPORADIC and the specified replenishment period is less than the initial budget.
EINVAL	The scheduling policy was SCHED_SPORADIC and the specified low priority is invalid.
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.
EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

1.4.18 pthread_exit

CALLING SEQUENCE:

```
#include <pthread.h>

void pthread_exit(
  void   *status
);
```

STATUS CODES:

NONE

DESCRIPTION:

NOTES:

1.4.19 pthread_detach

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_detach(
  pthread_t   thread
);
```

STATUS CODES:

ESRCH	The thread specified is invalid.
EINVAL	The thread specified is not a joinable thread.

DESCRIPTION:

NOTES:

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.

1.4.20 pthread_join

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_join(
  pthread_t   thread,
  void      **value_ptr
);
```

STATUS CODES:

ESRCH	The thread specified is invalid.
EINVAL	The thread specified is not a joinable thread.
EDEADLK	A deadlock was detected or thread is the calling thread.

DESCRIPTION:

NOTES:

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.

If value_ptr is NULL, then no value is returned.

1.4.21 pthread_self

CALLING SEQUENCE:

```
#include <pthread.h>

pthread_t pthread_self( void );
```

STATUS CODES:

This routine returns the id of the calling thread.

DESCRIPTION:

NOTES:

1.4.22 pthread_equal

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_equal(
  pthread_t  t1,
  pthread_t  t2
);
```

STATUS CODES:

zero	The thread ids are not equal.
non-zero	The thread ids are equal.

DESCRIPTION:

NOTES:

The behavior is undefined if the thread IDs are not valid.

1.4.23 pthread_once

CALLING SEQUENCE:

```
#include <pthread.h>

pthread_once_t once_control = PTHREAD_ONCE_INIT;

int pthread_once(
  pthread_once_t  *once_control,
  void           (*init_routine)(void)
);
```

STATUS CODES:

NONE

DESCRIPTION:

NOTES:

1.4.24 pthread_setschedparam

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_setschedparam(
  pthread_t           thread,
  int                 policy,
  struct sched_param *param
);
```

STATUS CODES:

EINVAL	The scheduling parameters indicated by the parameter param is invalid.
EINVAL	The value specified by policy is invalid.
EINVAL	The scheduling policy was SCHED_SPORADIC and the specified replenishment period is less than the initial budget.
EINVAL	The scheduling policy was SCHED_SPORADIC and the specified low priority is invalid.
ESRCH	The thread indicated was invalid.

DESCRIPTION:

NOTES:

1.4.25 pthread_getschedparam

CALLING SEQUENCE:

```
#include <pthread.h>

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

STATUS CODES:

EINVAL	The policy pointer argument is invalid.
EINVAL	The scheduling parameters pointer argument is invalid.
ESRCH	The thread indicated by the parameter thread is invalid.

DESCRIPTION:

NOTES:

2 Signal Manager

2.1 Introduction

The signal manager ...

The directives provided by the signal manager are:

- `sigaddset` -
- `sigdelset` -
- `sigfillset` -
- `sigismember` -
- `sigemptyset` -
- `sigaction` -
- `pthread_kill` -
- `sigprocmask` -
- `pthread_sigmask` -
- `kill` -
- `sigpending` -
- `sigsuspend` -
- `pause` -
- `sigwait` -
- `sigwaitinfo` -
- `sigtimedwait` -
- `sigqueue` -
- `alarm` -

2.2 Background

2.2.1 Signal Delivery

Signals directed at a thread are delivered to the specified thread.

Signals directed at a process are delivered to a thread which is selected based on the following algorithm:

1. If the action for this signal is currently SIG_IGN, then the signal is simply ignored.
2. If the currently executing thread has the signal unblocked, then the signal is delivered to it.

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.
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.
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.

2.3 Operations

2.4 Directives

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.

2.4.1 sigaddset

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.2 sigdelset

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.3 sigfillset

CALLING SEQUENCE:

```
#include <signal.h>

int sigfillset(
  sigset_t   *set
);
```

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.4 sigismember

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.5 sigemptyset

CALLING SEQUENCE:

```
#include <signal.h>

int sigemptyset(
  sigset_t   *set
);
```

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.6 sigaction

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINVAL	Invalid argument passed.
ENOTSUP	Realtime Signals Extension option not supported.

DESCRIPTION:

NOTES:

The signal number cannot be SIGKILL.

2.4.7 pthread_kill

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

ESRCH	The thread indicated by the parameter thread is invalid.
EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.8 sigprocmask

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.9 pthread_sigmask

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

2.4.10 kill

CALLING SEQUENCE:

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

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

STATUS CODES:

EINVAL	Invalid argument passed.
EPERM	Process does not have permission to send the signal to any receiving process.
ESRCH	The process indicated by the parameter pid is invalid.

DESCRIPTION:

NOTES:

2.4.11 sigpending

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

EFAULT	Invalid address for set.

DESCRIPTION:

NOTES:

2.4.12 sigsuspend

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

	Returns -1 and sets errno.
EINTR	Signal interrupted this function.

DESCRIPTION:

NOTES:

2.4.13 pause

CALLING SEQUENCE:

```
#include <signal.h>

int pause( void );
```

STATUS CODES:

	Returns -1 and sets errno.
EINTR	Signal interrupted this function.

DESCRIPTION:

NOTES:

2.4.14 sigwait

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINVAL	Invalid argument passed.
EINTR	Signal interrupted this function.

DESCRIPTION:

NOTES:

2.4.15 sigwaitinfo

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EINTR	Signal interrupted this function.

DESCRIPTION:

NOTES:

2.4.16 sigtimedwait

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

EAGAIN	Timed out while waiting for the specified signal set.
EINVAL	Nanoseconds field of the timeout argument is invalid.
EINTR	Signal interrupted this function.

DESCRIPTION:

NOTES:

If timeout is NULL, then the thread will wait forever for the specified signal set.

2.4.17 sigqueue

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

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.
EINVAL	The value of the signo argument is an invalid or unsupported signal number.
EPERM	The process does not have the appropriate privilege to send the signal to the receiving process.
ESRCH	The process pid does not exist.

DESCRIPTION:

NOTES:

2.4.18 alarm

CALLING SEQUENCE:

```
#include <signal.h>

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

STATUS CODES:

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.

DESCRIPTION:

NOTES:

3 Mutex Manager

3.1 Introduction

The mutex manager ...

The directives provided by the mutex manager are:

- `pthread_mutexattr_init` -
- `pthread_mutexattr_destroy` -
- `pthread_mutexattr_setprotocol` -
- `pthread_mutexattr_getprotocol` -
- `pthread_mutexattr_setprioceiling` -
- `pthread_mutexattr_getprioceiling` -
- `pthread_mutexattr_setpshared` -
- `pthread_mutexattr_getpshared` -
- `pthread_mutex_init` -
- `pthread_mutex_destroy` -
- `pthread_mutex_lock` -
- `pthread_mutex_trylock` -
- `pthread_mutex_timedlock` -
- `pthread_mutex_unlock` -
- `pthread_mutex_setprioceiling` -
- `pthread_mutex_getprioceiling` -

3.2 Background

3.3 Operations

3.4 Directives

This section details the mutex 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.

3.4.1 pthread_mutexattr_init

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_init(
  pthread_mutexattr_t *attr
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.

DESCRIPTION:

NOTES:

3.4.2 pthread_mutexattr_destroy

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_destroy(
  pthread_mutexattr_t *attr
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.

DESCRIPTION:

NOTES:

3.4.3 pthread_mutexattr_setprotocol

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_setprotocol(
  pthread_mutexattr_t   *attr,
  int                    protocol
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The protocol argument is invalid.

DESCRIPTION:

NOTES:

3.4.4 pthread_mutexattr_getprotocol

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_getprotocol(
  pthread_mutexattr_t   *attr,
  int                   *protocol
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The protocol pointer argument is invalid.

DESCRIPTION:

NOTES:

3.4.5 pthread_mutexattr_setprioceiling

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_setprioceiling(
  pthread_mutexattr_t   *attr,
  int                    prioceiling
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The prioceiling argument is invalid.

DESCRIPTION:

NOTES:

3.4.6 pthread_mutexattr_getprioceiling

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_getprioceiling(
  const pthread_mutexattr_t   *attr,
  int                         *prioceiling
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The prioceiling pointer argument is invalid.

DESCRIPTION:

NOTES:

3.4.7 pthread_mutexattr_setpshared

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_setpshared(
  pthread_mutexattr_t   *attr,
  int                    pshared
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The pshared argument is invalid.

DESCRIPTION:

NOTES:

3.4.8 pthread_mutexattr_getpshared

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutexattr_getpshared(
  const pthread_mutexattr_t   *attr,
  int                         *pshared
);
```

STATUS CODES:

EINVAL	The attribute pointer argument is invalid.
EINVAL	The attribute set is not initialized.
EINVAL	The pshared pointer argument is invalid.

DESCRIPTION:

NOTES:

3.4.9 pthread_mutex_init

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutex_init(
  pthread_mutex_t           *mutex,
  const pthread_mutexattr_t *attr
);
```

STATUS CODES:

EINVAL	The attribute set is not initialized.
EINVAL	The specified protocol is invalid.
EAGAIN	The system lacked the necessary resources to initialize another mutex.
ENOMEM	Insufficient memory exists to initialize the mutex.
EBUSY	Attempted to reinialize the object reference by mutex, a previously initialized, but not yet destroyed.

DESCRIPTION:

NOTES:

3.4.10 pthread_mutex_destroy

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutex_destroy(
  pthread_mutex_t           *mutex
);
```

STATUS CODES:

EINVAL	The specified mutex is invalid.
EBUSY	Attempted to destroy the object reference by mutex, while it is locked or referenced by another thread.

DESCRIPTION:

NOTES:

3.4.11 pthread_mutex_lock

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutex_lock(
  pthread_mutex_t           *mutex
);
```

STATUS CODES:

EINVAL	The specified mutex is invalid.
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.
EDEADLK	The current thread already owns the mutex.

DESCRIPTION:

NOTES:

3.4.12 pthread_mutex_trylock

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutex_trylock(
  pthread_mutex_t           *mutex
);
```

STATUS CODES:

EINVAL	The specified mutex is invalid.
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.
EDEADLK	The current thread already owns the mutex.

DESCRIPTION:

NOTES:

3.4.13 pthread_mutex_timedlock

CALLING SEQUENCE:

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

int pthread_mutex_timedlock(
  pthread_mutex_t           *mutex,
  const struct timespec     *timeout
);
```

STATUS CODES:

EINVAL	The specified mutex is invalid.
EINVAL	The nanoseconds field of timeout is invalid.
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.
EDEADLK	The current thread already owns the mutex.

DESCRIPTION:

NOTES:

3.4.14 pthread_mutex_unlock

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutex_unlock(
  pthread_mutex_t           *mutex
);
```

STATUS CODES:

EINVAL	The specified mutex is invalid.

DESCRIPTION:

NOTES:

3.4.15 pthread_mutex_setprioceiling

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutex_setprioceiling(
  pthread_mutex_t   *mutex,
  int                prioceiling,
  int               *oldceiling
);
```

STATUS CODES:

EINVAL	The oldceiling pointer parameter is invalid.
EINVAL	The prioceiling parameter is an invalid priority.
EINVAL	The specified mutex is invalid.

DESCRIPTION:

NOTES:

3.4.16 pthread_mutex_getprioceiling

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_mutex_getprioceiling(
  pthread_mutex_t   *mutex,
  int               *prioceiling
);
```

STATUS CODES:

EINVAL	The prioceiling pointer parameter is invalid.
EINVAL	The specified mutex is invalid.

DESCRIPTION:

NOTES:

4 Condition Variable Manager

4.1 Introduction

The condition variable manager ...

The directives provided by the condition variable manager are:

- `pthread_condattr_init` -
- `pthread_condattr_destroy` -
- `pthread_condattr_setpshared` -
- `pthread_condattr_getpshared` -
- `pthread_cond_init` -
- `pthread_cond_destroy` -
- `pthread_cond_signal` -
- `pthread_cond_broadcast` -
- `pthread_cond_wait` -
- `pthread_cond_timedwait` -

4.2 Background

4.3 Operations

4.4 Directives

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.

4.4.1 pthread_condattr_init

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_condattr_init(
  pthread_condattr_t *attr
);
```

STATUS CODES:

ENOMEM	Insufficient memory is available to initialize the condition variable attributes object.

DESCRIPTION:

NOTES:

4.4.2 pthread_condattr_destroy

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_condattr_destroy(
  pthread_condattr_t *attr
);
```

STATUS CODES:

EINVAL	The attribute object specified is invalid.

DESCRIPTION:

NOTES:

4.4.3 pthread_condattr_setpshared

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_condattr_setpshared(
  pthread_condattr_t   *attr,
  int                   pshared
);
```

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

4.4.4 pthread_condattr_getpshared

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_condattr_getpshared(
  const pthread_condattr_t   *attr,
  int                        *pshared
);
```

STATUS CODES:

EINVAL	Invalid argument passed.

DESCRIPTION:

NOTES:

4.4.5 pthread_cond_init

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_cond_init(
  pthread_cond_t           *cond,
  const pthread_condattr_t *attr
);
```

STATUS CODES:

EAGAIN	The system lacked a resource other than memory necessary to create the initialize the condition variable object.
ENOMEM	Insufficient memory is available to initialize the condition variable object.
EBUSY	The specified condition variable has already been initialized.
EINVAL	The specified attribute value is invalid.

DESCRIPTION:

NOTES:

4.4.6 pthread_cond_destroy

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_cond_destroy(
  pthread_cond_t           *cond
);
```

STATUS CODES:

EINVAL	The specified condition variable is invalid.
EBUSY	The specified condition variable is currently in use.

DESCRIPTION:

NOTES:

4.4.7 pthread_cond_signal

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_cond_signal(
  pthread_cond_t           *cond
);
```

STATUS CODES:

EINVAL	The specified condition variable is not valid.

DESCRIPTION:

NOTES:

This routine should not be invoked from a handler from an asynchronous signal handler or an interrupt service routine.

4.4.8 pthread_cond_broadcast

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_cond_broadcast(
  pthread_cond_t  *cond
);
```

STATUS CODES:

EINVAL	The specified condition variable is not valid.

DESCRIPTION:

NOTES:

This routine should not be invoked from a handler from an asynchronous signal handler or an interrupt service routine.

4.4.9 pthread_cond_wait

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_cond_wait(
  pthread_cond_t          *cond,
  pthread_mutex_t         *mutex
);
```

STATUS CODES:

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.

DESCRIPTION:

NOTES:

4.4.10 pthread_cond_timedwait

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_cond_timedwait(
  pthread_cond_t        *cond,
  pthread_mutex_t       *mutex,
  const struct timespec *abstime
);
```

STATUS CODES:

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.
ETIMEDOUT	The specified time has elapsed without the condition variable being satisfied.

DESCRIPTION:

NOTES:

5 Key Manager

5.1 Introduction

The key manager ...

The directives provided by the key manager are:

- `pthread_key_create` -
- `pthread_key_delete` -
- `pthread_setspecific` -
- `pthread_getspecific` -

5.2 Background

5.3 Operations

5.4 Directives

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.

5.4.1 pthread_key_create

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_key_create(
  pthread_key_t  *key,
  void          (*destructor)( void * )
);
```

STATUS CODES:

EAGAIN	There were not enough resources available to create another key.
ENOMEM	Insufficient memory exists to create the key.

5.4.2 pthread_key_delete

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_key_delete(
  pthread_key_t  key,
);
```

STATUS CODES:

EINVAL	The key was invalid

DESCRIPTION:

NOTES:

5.4.3 pthread_setspecific

CALLING SEQUENCE:

```
#include <pthread.h>

int pthread_setspecific(
  pthread_key_t  key,
  const void    *value
);
```

STATUS CODES:

EINVAL	The specified key is invalid.

DESCRIPTION:

NOTES:

5.4.4 pthread_getspecific

CALLING SEQUENCE:

```
#include <pthread.h>

void *pthread_getspecific(
  pthread_key_t  key
);
```

STATUS CODES:

NULL	There is no thread-specific data associated with the specified key.
non-NULL	The data associated with the specified key.

DESCRIPTION:

NOTES:

6 Clock Manager

6.1 Introduction

The clock manager ...

The directives provided by the clock manager are:

- `clock_gettime` -
- `clock_settime` -
- `clock_getres` -
- `nanosleep` -
- `time` -

6.2 Background

6.3 Operations

6.4 Directives

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.

6.4.1 clock_gettime

CALLING SEQUENCE:

```
#include <time.h>

int clock_gettime(
  clockid_t        clock_id,
  struct timespec *tp
);
```

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

EINVAL	The tp pointer parameter is invalid.
EINVAL	The clock_id specified is invalid.

DESCRIPTION:

NOTES:

6.4.2 clock_settime

CALLING SEQUENCE:

```
#include <time.h>

int clock_settime(
  clockid_t              clock_id,
  const struct timespec *tp
);
```

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

EINVAL	The tp pointer parameter is invalid.
EINVAL	The clock_id specified is invalid.
EINVAL	The contents of the tp structure are invalid.

DESCRIPTION:

NOTES:

6.4.3 clock_getres

CALLING SEQUENCE:

```
#include <time.h>

int clock_getres(
  clockid_t        clock_id,
  struct timespec *res
);
```

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

EINVAL	The res pointer parameter is invalid.
EINVAL	The clock_id specified is invalid.

DESCRIPTION:

NOTES:

If res is NULL, then the resolution is not returned.

6.4.4 sleep

CALLING SEQUENCE:

```
#include <time.h>

unsigned int sleep(
  unsigned int seconds
);
```

STATUS CODES:

This routine returns the number of unslept seconds.

DESCRIPTION:

NOTES:

This call is interruptible by a signal.

6.4.5 nanosleep

CALLING SEQUENCE:

```
#include <time.h>

int nanosleep(
  const struct timespec *rqtp,
  struct timespec       *rmtp
);
```

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

EINTR	The routine was interrupted by a signal.
EAGAIN	The requested sleep period specified negative seconds or nanoseconds.
EINVAL	The requested sleep period specified an invalid number for the nanoseconds field.

DESCRIPTION:

NOTES:

This call is interruptible by a signal.

6.4.6 nanosleep

CALLING SEQUENCE:

```
#include <time.h>

int time(
  time_t  *tloc
);
```

STATUS CODES:

This routine returns the number of seconds since the Epoch.

DESCRIPTION:

NOTES:

7 Scheduler Manager

7.1 Introduction

The scheduler manager ...

The directives provided by the scheduler manager are:

- `sched_get_priority_min` -
- `sched_get_priority_max` -
- `sched_rr_get_interval` -
- `sched_yield` -

7.2 Background

7.2.1 Priority

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.

7.2.2 Scheduling Policies

The following scheduling policies are available:

SCHED_FIFO	Priority-based, preemptive scheduling with no timeslicing. This is equivalent to what is called "manual round-robin" scheduling.
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_OTHER	Priority-based, preemptive scheduling with timeslicing. Time quantums are maintained on a per-thread basis and are reset at each context switch.
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.

7.3 Operations

7.4 Directives

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.

7.4.1 sched_get_priority_min

CALLING SEQUENCE:

```
#include <sched.h>

int sched_get_priority_min(
  int policy
);
```

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

EINVAL	The indicated policy is invalid.

DESCRIPTION:

NOTES:

7.4.2 sched_get_priority_max

CALLING SEQUENCE:

```
#include <sched.h>

int sched_get_priority_max(
  int policy
);
```

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

EINVAL	The indicated policy is invalid.

DESCRIPTION:

NOTES:

7.4.3 sched_rr_get_interval

CALLING SEQUENCE:

```
#include <sched.h>

int sched_rr_get_interval(
  pid_t            pid,
  struct timespec *interval
);
```

STATUS CODES:

On error, this routine returns -1 and sets errno to one of the following:

ESRCH	The indicated process id is invalid.
EINVAL	The specified interval pointer parameter is invalid.

DESCRIPTION:

NOTES:

7.4.4 sched_yield

CALLING SEQUENCE:

```
#include <sched.h>

int sched_yield( void );
```

STATUS CODES:

This routine always returns zero to indicate success.

DESCRIPTION:

NOTES:

Command and Variable Index

There are currently no Command and Variable Index entries.

Concept Index

There are currently no Concept Index entries.

Table of Contents