POSIX.1c/D10 Summary

Disclaimer

SunSoft, Inc.
2550 Garcia Avenue
Mountain View, California 94043

Introduction

All source that uses POSIX.1c threads must include the header file.

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

In addition, Solaris requires the pre-processor symbol **_REENTRANT** to be defined in the source code before any C source (including header files).

```
#define_REENTRANT
```

The POSIX.1c thread library should be the last library specified on the `cc(1)` command line.

```
voyager% cc -D_REENTRANT ... -lpthread
```

Name Space

Each POSIX.1c type is of the form:


```
pthread[_object]_t
```

Each POSIX.1c function has the form

```
pthread[_object]_operation[_np|_NP]
```

where *`object`* is a type (not required if object is a thread), `operation` is a type-specific operation and `np` (or `NP`) is used to identify non-portable, implementation specific functions.

All POSIX.1c functions (except for `pthread_exit`, `pthread_getspecific` and `pthread_self`) return zero (0) for success or an `errno` value if the operation fails.

There are eight(8) POSIX.1c types:

Table 0-1 POSIX.1c types

Type	Description
pthread_attr_t	Thread attribute
pthread_mutexattr_t	Mutual Exclusion Lock attribute
pthread_condattr_t	Condition variable attribute
pthread_mutex_t	Mutual Exclusion Lock (mutex)
pthread_cond_t	Condition variable (cv)
pthread_t	Thread ID
pthread_once_t	Once-only execution
pthread_key_t	Thread Specific Data (TSD) key

Feature Test Macros

POSIX.1c consists of a base (or common) component and a number of implementation optional components The base is the set of required operations to be supplied by every implementation. The pre-processor symbol (**_POSIX_THREADS**) can be used to test for the presence of the POSIX.1c base. Additionally, the standards document describes a set of six (6) optional components. A pre-processor symbol can be used to test for the presence of each All of the symbols appear in the following table.

Table 0-2 POSIX.1c Feature Test Macros

Feature Test Macro	Description
_POSIX_THREADS	base threads
_POSIX_THREAD_ATTR_STACKADDR	stack address attribute
_POSIX_THREAD_ATTR_STACKSIZE	stack size attribute
_POSIX_THREAD_PRIORITY_SCHEDULING	thread priority scheduling
_POSIX_THREAD_PRIO_INHERIT	mutex priority inheritance
_POSIX_THREAD_PRIO_PROTECT	mutex priority ceiling
_POSIX_THREAD_PROCESS_SHARED	inter-process synchronization

Macro Dependency

If **_POSIX_THREAD_PRIO_INHERIT** is defined then **_POSIX_THREAD_PRIORITY_SCHEDULING** is defined.

If **_POSIX_THREAD_PRIO_PROTECT** is defined then **_POSIX_THREAD_PRIORITY_SCHEDULING** is defined.

If **_POSIX_THREAD_PRIORITY_SCHEDULING** is defined then **_POSIX_THREADS** is defined.

If **_POSIX_THREADS** is defined then **_POSIX_THREAD_SAFE_FUNCTIONS** is defined.

POSIX.1c API

In the following sections, function arguments that are of the form:

```
type name = NULL
```

indicate that a value of **NULL** may safely be used for `name`.

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

Register functions to be called during fork execution.
errors **ENOMEM**
notes `prepare` functions are called in reverse order of registration.
`parent` and `child` functions are called in order of registration.

Thread Attributes

All thread attributes are set in an attribute object by a function of the form:

```
int    pthread_attr_setname( pthread_attr_t *attr, Type t );
```

All thread attributes are retrieved from an attribute object by a function of the form:

```
int    pthread_attr_getname( const pthread_attr_t *attr, Type *t );
```

Where *`name`* and *`Type`* are from the table below.

Table 0-3 Thread Attributes

Name and Type	Feature Test Macro	Value(s)
`int inheritsched`	**_POSIX_THREAD_PRIORITY_SCHEDULING**	**PTHREAD_INHERIT_SCHED,** **PTHREAD_EXPLICIT_SCHED**
`int schedpolicy`	**_POSIX_THREAD_PRIORITY_SCHEDULING**	**SCHED_FIFO,** **SCHED_RR,** **SCHED_OTHER**
`struct sched_param schedparam`	**_POSIX_THREADS**	POSIX.1b, Section 13
`int contentionscope`	**_POSIX_THREAD_PRIORITY_SCHEDULING**	**PTHREAD_SCOPE_SYSTEM,** **PTHREAD_SCOPE_PROCESS**
`size_t stacksize`	**_POSIX_THREAD_ATTR_STACKSIZE**	>= **PTHREAD_STACK_MIN**

Table 0-3 Thread Attributes

Name and Type	Feature Test Macro	Value(s)
`void *stackaddr`	**_POSIX_THREAD_ATTR_STACKADDR**	`void *stack`
`int detachstate`	**_POSIX_THREADS**	**PTHREAD_CREATE_DETACHED,** **PTHREAD_CREATE_JOINABLE**

```
int    pthread_attr_init( pthread_attr_t *attr );
```

Initialize a thread attribute object.
errors **ENOMEM**

```
int    pthread_attr_destroy( pthread_attr_t *attr );
```

Destroy a thread attribute object.
errors none

Thread Management

```
int    pthread_create( pthread_t *thread,
                       const pthread_attr_t *attr = NULL,
                       void *(*entry)(void *), void *arg );
```

Create a new thread of execution.
errors **EAGAIN**, **EINVAL**
note Maximum number of **PTHREAD_THREADS_MAX** threads per process.

```
int    pthread_detach( pthread_t thread );
```

Set the `detachstate` of the specified thread to **PTHREAD_CREATE_DETACHED**.
errors **EINVAL**, **ESRCH**

```
pthread_t    pthread_self( void );
```

Return the thread ID of the calling thread.
errors none

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

Compare two thread IDs for equality.
errors none

```
void   pthread_exit( void *status = NULL );
```

Terminate the calling thread.
errors none

```
int    pthread_join( pthread_t thread, void **status = NULL );
```

Synchronize with the termination of a thread.
errors **EINVAL**, **ESRCH**, **EDEADLK**
note This function is a cancellation point.

```
#include <sched.h>
int    pthread_getschedparam( pthread_t thread, int *policy, struct sched_param *param );
```

Get the scheduling policy and parameters of the specified thread.
control **_POSIX_THREAD_PRIORITY_SCHEDULING**
errors **ENOSYS**, **ESRCH**

```
#include <sched.h>
int    pthread_setschedparam( pthread_t thread, int policy,
                              const struct sched_param *param );
```

Set the scheduling policy and parameters of the specified thread.
control **_POSIX_THREAD_PRIORITY_SCHEDULING**
errors **ENOSYS**, **EINVAL**, **ENOTSUP**, **EPERM**, **ESRCH**
`policy` { **SCHED_RR**, **SCHED_FIFO**, **SCHED_OTHER** }

Mutex Attributes

All mutex attributes are set in a mutex attribute object by a function of the form:

```
int    pthread_mutexattr_setname( pthread_attr_t *attr, Type t );
```

All mutex attributes are retrieved from a mutex attribute object by a function of the form:

```
int    pthread_mutexattr_getname( const pthread_attr_t *attr, Type *t );
```

Where *name* and *Type* are from the table below

Table 0-4 Mutex Attributes

Name and Type	Feature Test Macro	Value(s)
`int protocol`	**_POSIX_THREAD_PRIO_INHERIT,**	**PTHREAD_PRIO_NONE,**
	_POSIX_THREAD_PRIO_PROTECT	**PTHREAD_PRIO_PROTECT,**
		PTHREAD_PRIO_INHERIT
`int pshared`	**_POSIX_THREAD_PROCESS_SHARED**	**PTHREAD_PROCESS_SHARED,**
		PTHREAD_PROCESS_PRIVATE
`int prioceiling`	**_POSIX_THREAD_PRIO_PROTECT**	POSIX.1b, Section 13

```
int    pthread_mutexattr_init( pthread_mutexattr_t *attr );
```
Initialize a mutex attribute object.
errors **ENOMEM**

```
int    pthread_mutexattr_destroy( pthread_mutexattr_t *attr );
```
Destroy a mutex attribute object.
errors **EINVAL**

Mutex Usage

```
int    pthread_mutex_init( pthread_mutex_t *mutex, const pthread_mutexattr_t *attr = NULL );
pthread_mutex_t     mutex        = PTHREAD_MUTEX_INITIALIZER;
```
Initialize a mutex.
errors **EAGAIN**, **ENOMEM**, **EPERM**, **EBUSY**, **EINVAL**

```
int    pthread_mutex_destroy( pthread_mutex_t *mutex );
```
Destroy a mutex.
errors **EBUSY**, **EINVAL**

```
int    pthread_mutex_getprioceiling( const pthread_mutex_t *mutex, int *prioceiling );
```
Get the `prioceiling` value of the specified mutex.
control **_POSIX_THREAD_PRIO_PROTECT**
errors **ENOSYS**, **EINVAL**, **EPERM**

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


Set the `prioceiling` value and return the old `prioceiling` value in the specified mutex.
control **_POSIX_THREAD_PRIO_PROTECT**
errors **ENOSYS**, **EINVAL**, **EPERM**

```
int    pthread_mutex_lock( pthread_mutex_t *mutex );
```
Acquire the indicated mutex.
errors **EINVAL**, **EDEADLK**

```
int    pthread_mutex_trylock( pthread_mutex_t *mutex );
```
Attempt to acquire the indicated mutex.
errors **EINVAL**, **EBUSY**, **EINVAL**

```
int    pthread_mutex_unlock( pthread_mutex_t *mutex );
```
Release the (previously acquired) mutex.
errors **EINVAL**, **EPERM**

Once-only Execution

```
pthread_once_t      once   = PTHREAD_ONCE_INIT;
```
Initialize a once control variable.

```
int    pthread_once( pthread_once_t *once_control, void (*init_routine)(void) );
```
Execute `init_routine` once.
errors none specified

Condition Variable Attributes

All condition variable attributes are set in a condition variable attribute object by a function of the form:

```
int    pthread_condattr_setname( pthread_condattr_t *attr, Type t );
```

All condition variable attributes are retreived from a condition variable attribute object by a function of the form:

```
int    pthread_condattr_getname( const pthread_condattr_t *attr, Type *t );
```

Where *name* and *Type* are from the table below

Table 0-5 Condition Variable Attributes

Name and Type	Feature Test Macro	Value(s)
`int pshared`	**_POSIX_THREAD_PROCESS_SHARED**	**PTHREAD_PROCESS_SHARED,**
		PTHREAD_PROCESS_PRIVATE

```
int    pthread_condattr_init( pthread_condattr_t *attr );
```
Initialize a condition variable attribute object.
errors **ENOMEM**

```
int    pthread_condattr_destroy( pthread_condattr_t *attr );
```
Destroy a condition variable attribute object.
errors **EINVAL**

Condition Variable Usage

```
int    pthread_cond_init( pthread_cond_t *cond,
```

```
                        const pthread_condattr_t *attr = NULL );
pthread_cond_t      cond        = PTHREAD_COND_INITIALIZER;
```

Initialize a condition variable.
errors **EAGAIN**, **ENOMEM**, **EBUSY**, **EINVAL**

```
int    pthread_cond_destroy( pthread_cond_t *cond );
```

Destroy a condition variable.
errors **EBUSY**, **EINVAL**

```
int    pthread_cond_signal( pthread_cond_t *cond );
```

Unblock at least one thread currently blocked in the specified condition variable.
errors **EINVAL**

```
int    pthread_cond_broadcast( pthread_cond_t *cond );
```

Unblock all threads currently blocked on the specified condition variable.
errors **EINVAL**

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

Block on the specified condition variable.
errors **EINVAL**
note This function is a cancellation point.

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

Block on the specified condition variable not longer than the specified absolute time.
errors **ETIMEDOUT**, **EINVAL**
note This function is a cancellation point.

Thread Specific Data

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

Create a thread-specific data key.
errors **EAGAIN**, **ENOMEM**
note system limit of **PTHREAD_KEYS_MAX** per process.
system limit of **PTHREAD_DESTRUCTOR_ITERATIONS** calls to `destructor` per thread exit.

```
int    pthread_key_delete( pthread_key_t key );
```

Destroy a thread-specific data key.
errors **EINVAL**

```
void   *pthread_getspecific( pthread_key_t key );
```

Return the value bound to the given key for the calling thread.
errors none

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

Set the value for the given key in the calling thread.
errors **ENOMEM**, **EINVAL**

Signal Management

```
#include <signal.h>
int    pthread_sigmask( int how, const sigset_t *newmask = NULL, sigset_t *oldmask = NULL );
```

Examine or change calling threads signal mask.

errors **EINVAL**
`how` { **SIG_BLOCK**, **SIG_UNBLOCK**, **SIG_SETMASK** }

```
#include <signal.h>
int    pthread_kill( pthread_t thread, int signo );
```

Deliver signal to indicated thread.
errors **ESRCH**, **EINVAL**

```
#include <signal.h>
int    sigwait( const sigset_t *set, int *sig );
```

Synchronously accept a signal.
errors **EINVAL**, **EINTR**
note This function is a cancellation point.

Cancellation

```
int    pthread_setcancelstate( int state, int *oldstate );
```

Set the cancellation state for the calling thread.
errors **EINVAL**
`state` { **PTHREAD_CANCEL_ENABLE**, **PTHREAD_CANCEL_DISABLE** }

```
int    pthread_setcanceltype( int type, int *oldtype );
```

Set the cancellation type for the calling thread.
errors **EINVAL**
`type` { **PTHREAD_CANCEL_DEFERRED**, **PTHREAD_CANCEL_ASYNCHRONOUS** }

```
int    pthread_cancel( pthread_t thread );
```

Cancel the specified thread.
errors **ESRCH**
note threads that have been cancelled terminate with a status of **PTHREAD_CANCELED.**

```
void   pthread_testcancel( void );
```

Introduce a cancellation point.
errors none
note This function is a cancellation point.

```
void   pthread_cleanup_pop( int execute );
```

Pop the top item from the cancellation stack and optionally execute it.
errors none specified
note push and pop operations must appear at the same lexical level.
`execute` { 1, 0 }

```
void   pthread_cleanup_push( void (*routine)(void *), void *arg );
```

Push an item onto the cancellation stack.
errors none specified