POSIX.1c/D10 Summary

Disclaimer

Copyright (C) 1995 by Sun Microsystems, Inc.

All rights reserved.

This file is a product of SunSoft, Inc. and is provided for unrestricted use provided that this legend is included on all media and as a part of the software program in whole or part. Users may copy, modify or distribute this file at will.

THIS FILE IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE WARRANTIES OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.

This file is provided with no support and without any obligation on the part of SunSoft, Inc. to assist in its use, correction, modification or enhancement.

SunSoft and Sun Microsystems, Inc. shall have no liability with respect to the infringement of copyrights, trade secrets or any patents by this file or any part thereof.

In no event will SunSoft or Sun Microsystems, Inc. be liable for any lost revenue or profits or other special, indirect and consequential damages, even if they have been advised of the possibility of such damages.

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]_][_operation]([_np|_NP])
```

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>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread_attr_t</td>
<td>Thread attribute</td>
</tr>
<tr>
<td>pthread_mutexattr_t</td>
<td>Mutual Exclusion Lock attribute</td>
</tr>
<tr>
<td>pthread_condattr_t</td>
<td>Condition variable attribute</td>
</tr>
<tr>
<td>pthread_mutex_t</td>
<td>Mutual Exclusion Lock (mutex)</td>
</tr>
<tr>
<td>pthread_cond_t</td>
<td>Condition variable (cv)</td>
</tr>
<tr>
<td>pthread_t</td>
<td>Thread ID</td>
</tr>
<tr>
<td>pthread_once_t</td>
<td>Once-only execution</td>
</tr>
<tr>
<td>pthread_key_t</td>
<td>Thread Specific Data (TSD) key</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Feature Test Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_POSIX_THREADS</td>
<td>base threads</td>
</tr>
<tr>
<td>_POSIX_THREAD_ATTR_STACKADDR</td>
<td>stack address attribute</td>
</tr>
<tr>
<td>_POSIX_THREAD_ATTR_STACKSIZE</td>
<td>stack size attribute</td>
</tr>
<tr>
<td>_POSIX_THREAD_PRIORITY_SCHEDULING</td>
<td>thread priority scheduling</td>
</tr>
<tr>
<td>_POSIX_THREAD_PRIO_INHERIT</td>
<td>mutex priority inheritance</td>
</tr>
<tr>
<td>_POSIX_THREAD_PRIO_PROTECT</td>
<td>mutex priority ceiling</td>
</tr>
<tr>
<td>_POSIX_THREAD_PROCESS_SHARED</td>
<td>inter-process synchronization</td>
</tr>
</tbody>
</table>

Macro Dependency

If `_POSIX_THREAD_PRIO_INHERIT` is defined then `_POSIX_THREAD_PRIORITY_SCHEDULING` is defined.
If `_POSIX_THREAD_PRIO_PROTECT` is defined then
_the _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.

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

Register functions to be called during fork execution.

enes

```
struct sched_paramschedparam
```

Where name and Type are from the table below:

<table>
<thead>
<tr>
<th>Name and Type</th>
<th>Feature Test Macros</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inheritsched</td>
<td>_POSIX_THREAD_PRIORITY_SCHEDULING</td>
<td>PTHREAD_INHERIT_SCHED, PTHREAD_EXPLICIT_SCHED</td>
</tr>
<tr>
<td>schedpolicy</td>
<td>_POSIX_THREAD_PRIORITY_SCHEDULING</td>
<td>SCHED_FIFO, SCHED_RR, SCHED_OTHER</td>
</tr>
<tr>
<td>schedparam</td>
<td>_POSIX_THREADS</td>
<td>POSIX 1b, Section 13</td>
</tr>
<tr>
<td>contentionscope</td>
<td>_POSIX_THREAD_PRIORITY_SCHEDULING</td>
<td>PTHREAD_SCOPE_SYSTEM, PTHREAD_SCOPE_PROCESS</td>
</tr>
<tr>
<td>size_t stacksize</td>
<td>_POSIX_THREAD_ATTR_STACKSIZE</td>
<td>&gt;= PTHREAD_STACK_MIN</td>
</tr>
</tbody>
</table>

Thread Attributes

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

```c
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:

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

Where name and Type are from the table below:

<table>
<thead>
<tr>
<th>Name and Type</th>
<th>Feature Test Macros</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *stackaddr</td>
<td>_POSIX_THREAD_ATTR_STACKADDR</td>
<td>void *stack</td>
</tr>
<tr>
<td>int detachstate</td>
<td>_POSIX_THREADS</td>
<td>PTHREAD_CREATE_DETACHED, PTHREAD_CREATE_JOINABLE</td>
</tr>
<tr>
<td>int pthread_attr_init( pthread_attr_t *attr );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int pthread_attr_destroy( pthread_attr_t *attr );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int pthread_create( pthread_t *thread, const pthread_attr_t *attr = NULL, void *(*entry)(void *), void *arg );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int pthread_detach( pthread_t thread );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pthread_t pthread_self( void );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int pthread_equal( pthread_t t1, pthread_t t2 );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int pthread_join( pthread_t thread, void **status = NULL );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void *stackaddr</td>
<td>_POSIX_THREAD_ATTR_STACKADDR</td>
<td>void *stack</td>
</tr>
<tr>
<td>int detachstate</td>
<td>_POSIX_THREADS</td>
<td>PTHREAD_CREATE_DETACHED, PTHREAD_CREATE_JOINABLE</td>
</tr>
</tbody>
</table>

Thread Management

Create a new thread of execution.

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

Set the detachstate of the specified thread to PTHREAD_CREATE_DETACHED

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

Return the thread ID of the calling thread.

```c
int pthread_self( void );
```

Compare two thread IDs for equality.

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

Terminate the calling thread.

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

Synchronize with the termination of a thread.

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

Get the scheduling policy and parameters of the specified thread.

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

Get the scheduling policy and parameters of the specified thread.

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

Control the _POSIX_THREAD_PRIORITY_SCHEDULING_ error

```c
#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.
__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_mutexattr_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_mutexattr_t *attr, Type t );

Where name and Type are from the table below

<table>
<thead>
<tr>
<th>Name and Type</th>
<th>Feature Test Macro</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int protocol</td>
<td>_POSIX_THREAD_PRIO_INHERIT, _POSIX_THREAD_PRIO_PROTECT</td>
<td>PTHREAD_PRIO_NONE, PTHREAD_PRIO_PROTECT, PTHREAD_PRIO_INHERIT</td>
</tr>
<tr>
<td>int pshared</td>
<td>_POSIX_THREAD_PROCESS_SHARED</td>
<td>PTHREAD_PROCESS_SHARED, PTHREAD_PROCESS_PRIVATE</td>
</tr>
<tr>
<td>int prioceiling</td>
<td>_POSIX_THREAD_PRIO_PROTECT</td>
<td>POSIX.1b, Section 13</td>
</tr>
</tbody>
</table>

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 );
Initialize a mutex.

Errors
EINVAL, ENOMEM, EPERM, EBUSY, EINVAL, EAGAIN

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.

Errors
ENOMEM, EINVAL, EPERM

int pthread_mutex_setprioceiling( pthread_mutex_t *mutex, int prioceiling, int *old_prioceiling );
Set the prioceiling value and return the old prioceiling value in the specified mutex.

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
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 retrieved 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>
<thead>
<tr>
<th>Name and Type</th>
<th>Feature Test Macro</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int pshared</td>
<td>_POSIX_THREAD_PROCESS_SHARED</td>
<td>PTHREAD_PROCESS_SHARED, PTHREAD_PROCESS_PRIVATE</td>
</tr>
</tbody>
</table>

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 );
Initialize a condition variable.

Errors
EINVAL, ENOMEM, EPERM

int pthread_cond_destroy( pthread_cond_t *cond );
Destroy a condition variable.

Errors
EINVAL, ENOMEM, EPERM
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

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.
note 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, EINVAL

#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

#include <signal.h>
int pthread_setcancelstate( int state, int *oldstate );
Set the cancellation state for the calling thread.
errors EINVAL
state {PTHREAD_CANCEL_ENABLE, PTHREAD_CANCEL_DISABLE}

#include <signal.h>
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