NAME

threads, pthreads, libpthread, libthread – thread libraries: libpthread and libthread

SYNOPSIS

POSIX

cc [*flag* ...] *file* ... **–lpthread** [**-lposix4** *library* ...]
#include <pthread.h>

Solaris

cc [*flag* ...] *file* ... **–lthread** [*library* ...]
#include <thread.h>
#include <sched.h>

MT-LEVEL

Fork1-Safe MT-Safe

DESCRIPTION

Two threads libraries are available, POSIX and Solaris. Both implementations are interoperable, their functionality similar, and can be used within the same application. However, only POSIX threads are guaranteed to be fully portable to other POSIX-compliant environments. As indicated by the "Synopsis" section above, their use requires different source include files and different linking libraries.

Similarities

Most of the functions in both libraries, **libpthread** and **libthread**, have a counterpart in the other's library. POSIX functions and Solaris functions, whose names have similar endings, usually have similar functionality, number of arguments, and use of arguments. i.e.:

POSIX	Solaris
pthread_kill( )	**thr_kill( )**
pthread_sigmask( )	**thr_sigsetmask( )**
pthread_mutex_lock( )	**mutex_lock( )**
sem_wait( )	**sema_wait( )**

All POSIX threads function names begin with the prefix "**pthread**", with semaphore names being the exception.

Differences

POSIX

- is more portable,
- establishes characteristics for each thread according to configurable attribute objects,
- implements thread cancellation,
- enforces scheduling algorithms, and
- allows for clean-up handlers for **fork**(2) calls.

Solaris

- threads can be suspended and continued,
- implements an optimized mutex, reader/writer locking.
- may increase the concurrency, and

- implements daemon threads, for whose demise the process does not wait.

IMPLEMENTATION

	POSIX	Solaris
Creation	**pthread_create( )**	**thr_create( )**
	pthread_attr_init( )	——
	pthread_attr_setdetachstate( )	——
	pthread_attr_getdetachstate( )	——
	pthread_attr_setinheritsched( )	——
	pthread_attr_getinheritsched( )	——
	pthread_attr_setschedparam( )	——
	pthread_attr_getschedparam( )	——
	pthread_attr_setschedpolicy( )	——
	pthread_attr_getschedpolicy( )	——
	pthread_attr_setscope( )	——
	pthread_attr_getscope( )	——
	pthread_attr_setstackaddr( )	——
	pthread_attr_getstackaddr( )	——
	pthread_attr_setstacksize( )	——
	pthread_attr_getstacksize( )	——
	pthread_attr_destroy( )	——
	——	**thr_min_stack( )**
Exit	**pthread_exit( )**	**thr_exit( )**
	pthread_join( )	**thr_join( )**
	pthread_detach( )	——
Thread Specific Data	**pthread_key_create( )**	**thr_keycreate( )**
	pthread_setspecific( )	**thr_setspecific( )**
	pthread_getspecific( )	**thr_getspecific( )**
	pthread_key_delete( )	——
Signal	**pthread_sigmask( )**	**thr_sigsetmask( )**
	pthread_kill( )	**thr_kill( )**
ID	**pthread_self( )**	**thr_self( )**
	pthread_equal( )	——
	——	**thr_main( )**
Scheduling	——	**thr_yield( )**
	——	**thr_suspend( )**
	——	**thr_continue( )**
	——	**thr_setconcurrency( )**
	——	**thr_getconcurrency( )**
	pthread_setschedparam( )	**thr_setprio( )**
	pthread_getschedparam( )	**thr_getprio( )**

Cancellation	**pthread_cancel( )**	——
	pthread_setcancelstate( )	——
	pthread_setcanceltype( )	——
	pthread_testcancel( )	——
	pthread_cleanup_pop( )	——
	pthread_cleanup_push( )	——
Mutex	**pthread_mutex_init( )**	**mutex_init( )**
	pthread_mutexattr_init( )	——
	pthread_mutexattr_setpshared( )	——
	pthread_mutexattr_getpshared( )	——
	pthread_mutexattr_setprotocol( )	——
	pthread_mutexattr_getprotocol( )	——
	pthread_mutexattr_setprioceiling( )	——
	pthread_mutexattr_getprioceiling( )	——
	pthread_mutexattr_destroy( )	——
	pthread_mutex_setprioceiling( )	——
	pthread_mutex_getprioceiling( )	——
	pthread_mutex_lock( )	**mutex_lock( )**
	pthread_mutex_trylock( )	**mutex_trylock( )**
	pthread_mutex_unlock( )	**mutex_unlock( )**
	pthread_mutex_destroy( )	**mutex_destroy( )**
Condition Variable	**pthread_cond_init( )**	**cond_init( )**
	pthread_condattr_init( )	——
	pthread_condattr_setpshared( )	——
	pthread_condattr_getpshared( )	——
	pthread_condattr_destroy( )	——
	pthread_cond_wait( )	**cond_wait( )**
	pthread_cond_timedwait( )	**cond_timedwait( )**
	pthread_cond_signal( )	**cond_signal( )**
	pthread_cond_broadcast( )	**cond_broadcast( )**
	pthread_cond_destroy( )	**cond_destroy( )**
Reader/Writer	——	**rwlock_init( )**
Locking	——	**rw_rdlock( )**
	——	**rw_tryrdlock( )**
	——	**rw_wrlock( )**
	——	**rw_trywrlock( )**
	——	**rw_unlock( )**
	——	**rwlock_destroy( )**
Semaphore	**sem_init( )**	**sema_init( )**
	sem_open( )	——
	sem_close( )	——
	sem_wait( )	**sema_wait( )**

	sem_trywait( )	**sema_trywait( )**
	sem_post( )	**sema_post( )**
	sem_getvalue( )	–––
	sem_unlink( )	–––
	sem_destroy( )	**sema_destroy( )**
fork() Clean Up Handling	**pthread_atfork( )**	–––
Limits	**pthread_once( )**	–––
Debugging	–––	**thr_stksegment( )**

LOCKING

Synchronization

Multi-threaded behavior is asynchronous, and therefore, optimized for concurrent and parallel processing. Since threads, always from within the same process and sometimes from multiple processes, share global data with each other, they are not guaranteed exclusive access to the shared data at any point in time. Securing mutually exclusive access to shared data requires synchronization among the threads. Solaris implements four synchronization mechanisms:

- mutex
- condition variable
- reader/writer locking *(optimized frequent-read occasional-write mutex)*
- semaphore

POSIX implements all but reader/writer locking.

Synchronizing multiple threads diminishes their concurrency. The coarser the grain of synchronization, that is, the larger the block of code that is locked, the lesser the concurrency.

MT fork()

If a multi-threaded program calls **fork**(2), it implicitly calls **fork1**(2), which replicates only the calling thread. Should there be any outstanding mutexes throughout the process, the application should call **pthread_atfork**(3T), to wait for and acquire those mutexes, prior to calling **fork( )**.

FILES

POSIX

/usr/include/pthread.h
/lib/libpthread.*
/lib/libposix4.*

Solaris

/usr/include/thread.h
/usr/include/sched.h
/lib/libthread.*

SEE ALSO

fork(2), **intro**(3), **pthread_atfork**(3T), **pthread_create**(3T)

ERRORS

In a multi-threaded application, linked with **libpthread** or **libthread**, **EINTR** may be returned whenever another thread calls **fork**(2), which calls **fork1**(2) instead.