**NAME**

pthread_setschedparam, pthread_getschedparam, thr_setprio, thr_getprio – dynamic access to thread scheduling

**SYNOPSIS**

**POSIX**

**cc** [ *flag* ... ] *file* ... **–lpthread** [ *library* ... ]
**#include <pthread.h>**
**int pthread_setschedparam(pthread_t** *target_thread,* **int** *policy,*
  **const struct sched_param** **param***);**
**int pthread_getschedparam(pthread_t** *target_thread,* **int** **policy,*
  **struct sched_param** **param***);**

**Solaris**

**cc** [ *flag* ... ] *file* ... **–lthread** [ *library* ... ]
**#include <thread.h>**
**int thr_setprio(thread_t** *target_thread,* **int** *priority***);**
**int thr_getprio(thread_t** *target_thread,* **int** **priority***);**

**MT-LEVEL**

MT-Safe

**DESCRIPTION**

Thread scheduling is controlled by three attributes: its scope of contention, being either inter-process or intra-process (bound vs. unbound), (see **priocntl**(2)); a relative scheduling priority; and a scheduling policy.

**Contentionscope**

Bound threads, which are inter-process, compete system-wide for scheduling resources and must be set at creation, for example:

**pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);**
**pthread_create(NULL, &attr, thread_routine, arg);**

OR

**thr_create(NULL, NULL, thread_routine, arg, THR_BOUND, NULL);**

A bound thread is bound to an LWP and its scheduling is dependent upon the scheduling of the LWP to which it is bound. LWPs compete with other LWPs in other processes, however, their scheduling may be dynamically controlled by **priocntl**(2), or **sched_setscheduler**(3R).

By default, the scope for newly-created threads are unbound, or intra-process, and their setting is **PTHREAD_SCOPE_PROCESS** or **NULL.** An unbound thread is scheduled by **libthread** or **libpthread** on an underlying LWP, which competes with other LWPs in the same process.

The following dynamic scheduling functions should be used only with unbound threads: **pthread_setschedparam( )**, **pthread_getschedparam( )**, **thr_setprio( )**, and **thr_getprio( ).**

**Priority**

Priority scheduling is determined as follows:

- Higher priority threads are scheduled before lower priority threads.
- Both POSIX and Solaris assume that the priority is inherited across a thread

create.

- POSIX can modify priority at creation time (see **pthread_attr_setschedparam**(3T)). Equivalently, a Solaris thread can be created suspended and its priority can be modified.

**pthread_setschedparam( )** and **thr_setprio( )** can dynamically modify an unbound thread's priority, and **pthread_getschedparam( )** and **thr_getprio( )** can read an unbound thread's priority.

**Policy**

The scheduling *policy* setting is:

**SCHED_OTHER** *(system default, often time-sharing)*
Competing threads in this class are multiplexed according to their relative *priority*.

NOTE: POSIX specifies, under an option, the additional policies, SCHED_FIFO and SCHED_RR . Solaris has chosen to not implement these options at this time. Equivalent functionality may be obtained by creating bound threads (i.e., threads with the PTHREAD_SCOPE_SYSTEM value for the *contentionscope* attribute), which use **priocntl**(2). See **pthread_create**(3T) and **priocntl**(2).

**POSIX Scheduling**

The **pthread_setschedparam( )** and **pthread_getschedparam( )** functions allow the scheduling policy and scheduling priority parameters to be retrieved and set for individual threads within a multi-threaded process.

The **pthread_setschedparam( )** function sets the scheduling policy and related scheduling priority for the thread ID given by *target_thread* to the policy and associated priority provided in *policy*, and the **sched_priority** member of *param*, respectively.

No scheduling parameters are changed for the target thread if **pthread_setschedparam( )** fails.

For **SCHED_OTHER**, the affected scheduling parameter is the *sched_priority* member of the **sched_param** structure.

Presently, **SCHED_OTHER** is the only policy supported. An **ENOSUP** error will occur following an attempt to set policy as **SCHED_FIFO** or **SCHED_RR.** (The latter two policies are optional under POSIX.)

The **pthread_getschedparam( )** function retrieves the scheduling policy and scheduling priority parameters for the thread ID given by *target_thread*, and then stores the values in *policy* and the **sched_priority** member of *param*, respectively.

**Solaris Scheduling**

Solaris scheduling may only dynamically affect *priority*. There is no functionality to alter the *policy* of any thread; by default, a Solaris thread's schedule is equivalent to **SCHED_OTHER,** which is the only available Solaris policy.

**thr_setprio( )** changes the priority of the thread, specified by *target_thread*, within the current process to the priority specified by *priority*. Currently, by default, threads are scheduled based on fixed priorities that range from zero, the least significant, to 127. The *target_thread* will preempt lower priority threads, and will yield to higher priority threads in their contention for LWPs, not CPUs.

The function **thr_getprio( )** stores the current priority for the thread specified by *target_thread* in the location pointed to by *priority*. Note that thread priorities regulate access to LWPs, not CPUs, and hence are different from real-time priorities, which regulate and enforce access to CPU resources. A thread's priority set via these functions is more like a hint in terms of guaranteed access to execution resources. Programs that need access to "real" priorities should use bound threads in the real-time class (see **priocntl**(2)).

**RETURN VALUES**

Zero is returned upon successful completion; otherwise, an error number is returned.

**ERRORS**

For each of the following conditions, these functions return an error number if the condition is detected.

**ESRCH** The value specified by *target_thread* does not refer to an existing thread.

For each of the following conditions, **pthread_setschedparam( )** and **pthread_getschedparam( )** return an error number if the condition is detected.

**ENOSUP** The only policy supported is **SCHED_OTHER.** Attempts to set policy as **SCHED_FIFO** or **SCHED_RR** will result in the error **ENOSUP.**

**EINVAL** The *policy* or *param* specified value is invalid.

For each of the following conditions, if the condition is detected, **thr_setprio( )** returns an error number.

**EINVAL** The value of *priority* makes no sense for the scheduling class associated with the *target_thread.*

**SEE ALSO**

**priocntl**(2), **sched_setparam**(3R), **sched_setscheduler**(3R), **pthread_attr_init**(3T), **pthread_create**(3T), **thr_suspend**(3T), **thr_yield**(3T)

**NOTES**

Currently, the only supported policy is **SCHED_OTHER**. Attempts to set policy as **SCHED_FIFO** or **SCHED_RR** will result in the error **ENOSUP.**