



Operating Systems CS F372

Threads

BIJU K RAVEENDRAN

Attributes

- `extern int pthread_attr_init (pthread_attr_t *attr)`
- `extern int pthread_attr_setdetachstate (pthread_attr_t *attr, int detachstate)`
- `extern int pthread_attr_getdetachstate (__const pthread_attr_t *attr, int *detachstate)`
- `extern int pthread_attr_setschedpolicy (pthread_attr_t *attr, int policy)`
- `extern int pthread_attr_getschedpolicy (__const pthread_attr_t *attr, int *policy)`

Sched Param

- Contain the scheduling parameters (essentially, the scheduling priority) for the thread.
- See sched_setparam for more information on scheduling parameters.
- Default value: priority is 0.
- This attribute is not significant if the scheduling policy is SCHED_OTHER; it only matters for the realtime policies SCHED_RR and SCHED_FIFO.
- The scheduling priority of a thread can be changed after creation with pthread_setschedparam



Sched Param

- `extern int pthread_attr_setschedparam(pthread_attr_t *attr, __const struct sched_param *param)`
- `extern int pthread_attr_getschedparam(__const pthread_attr_t *attr, struct sched_param *param)`
- See struct sched_param in bits/sched.h

Inheritsched

- Indicate whether the scheduling policy and scheduling parameters for the newly created thread are determined by the values of the schedpolicy and schedparam attributes (value PTHREAD_EXPLICIT_SCHED) or are inherited from the parent thread (value PTHREAD_INHERIT_SCHED).
- Default value: PTHREAD_EXPLICIT_SCHED.
- extern int pthread_attr_setinheritsched (pthread_attr_t *attr, int inherit)
- extern int pthread_attr_getinheritsched (__const pthread_attr_t *attr, int *inherit)

Scope

- Define the scheduling contention scope for the created thread.
- The only value supported in the LinuxThreads implementation is `PTHREAD_SCOPE_SYSTEM`
 - meaning that the threads contend for CPU time with all processes running on the machine (thread priorities are interpreted relative to the priorities of all other processes on the machine).
- The other value specified by the standard, `PTHREAD_SCOPE_PROCESS`
 - means that scheduling contention occurs only between the threads of the running process (thread priorities are interpreted relative to the priorities of the other threads of the process, regardless of the priorities of other processes)

Scope

- `extern int pthread_attr_setscope (pthread_attr_t *attr, int scope)`
- `extern int pthread_attr_getscope (__const pthread_attr_t *attr, int *scope)`

SetSchedParam

- `extern int pthread_setschedparam (pthread_t t_thread, int policy, __const struct sched_param *param)`
 - sets the scheduling parameters for the thread `t_thread` as indicated by `policy` and `param`.
 - Policy can be either `SCHED_OTHER`, `SCHED_RR` or `SCHED_FIFO`.
 - `param` specifies the scheduling priority for the two realtime policies.
- `extern int pthread_getschedparam (pthread_t t_thread, int *policy, struct sched_param *param)`
 - retrieves the scheduling policy and scheduling parameters for the thread `t_thread` and store them in the locations pointed to by `policy` and `param`, respectively.
- Return value
 - return 0 on success
 - a non-zero error code on error.

Self & equal

- extern pthread_t pthread_self (void)
 - return the thread identifier for the calling thread.
- extern int pthread_equal (pthread_t __thread1, pthread_t __thread2)
 - determines if two thread identifiers refer to the same thread.
 - Returns a non-zero value if thread1 and thread2 refer to the same thread. Otherwise, 0 is returned

Detach

- `extern int pthread_detach (pthread_t th)`
 - put the thread `th` in the detached state.
 - applies to threads created in the joinable state, and which needs to be put in the detached state later.
 - After `pthread_detach` completes, subsequent attempts to perform `pthread_join` on `th` will fail.
 - If another thread is already joining the thread `th` at the time `pthread_detach` is called, `pthread_detach` does nothing and leaves `th` in the joinable state.
- Return value
 - On success, 0 is returned.
 - On error, a non-zero error code is returned.

Exit

- `extern void pthread_exit (void *retval)`
 - terminates the execution of the calling thread.
 - All cleanup handlers that have been set for the calling thread with `pthread_cleanup_push` are executed in reverse order.
 - Finalization functions for thread-specific data are then called for all keys that have non- NULL values associated with them in the calling thread (see `pthread_key_create`).
 - Finally, execution of the calling thread is stopped.
 - The `retval` argument is the return value of the thread. It can be consulted from another thread using `pthread_join`.
- Return value
 - The `pthread_exit` function never returns.

Join

- `extern int pthread_join (pthread_t th, void ** __thread_return)`
 - suspends the execution of the calling thread until the thread identified by `th` terminates, either by calling `pthread_exit` or by being cancelled.
 - If `thread_return` is not `NULL`, the return value of `th` is stored in the location pointed to by `thread_return`.
 - The return value of `th` is either the argument it gave to `pthread_exit`, or `PTHREAD_CANCELED` if `th` was cancelled.
 - The joined thread `th` must be in the joinable state
 - When a joinable thread terminates, its memory resources (thread descriptor and stack) are not deallocated until another thread performs `pthread_join` on it.
 - It is must to call `pthread_join` once for each joinable thread created to avoid memory leaks.

Join

- At most one thread can wait for the termination of a given thread.
- Calling `pthread_join` on a thread `th` on which another thread is already waiting for termination returns an error.
- Cancellation
 - `pthread_join` is a cancellation point.
 - If a thread is canceled while suspended in `pthread_join`, the thread execution resumes immediately and the cancellation is executed without waiting for the `th` thread to terminate.
 - If cancellation occurs during `pthread_join`, the `th` thread remains not joined.
- Return value
 - On success, the return value of `th` is stored in the location pointed to by `thread_return`, and 0 is returned.
 - On error, a non-zero error code is returned.

