OS Tutorial : Thread

Huan Wang
huanwang@uvic.ca

Outline

* **Pthreads API**
  * Thread Creation, Attributes & Termination
  * Sample Codes
* **Thread Synchronization**
  * Mutual Exclusions (Mutex)
  * Condition Variables (Convar)

Mutex Variable (1)

* What is a mutex?
  * A basic mechanism supplied by the pthreads library to assure **exclusive access** to variables (or **critical sections)**.
  * Can be imagined as a **lock** to the critical section, so that only one thread can get access to the critical section at the same time.
  * If a thread attempts to lock a mutex that was locked by another thread,
    * the thread will be **suspended**;
    * after been suspended, the thread will **not** consume CPU resources;
    * the thread resumes when the mutex is **unlocked**.
  * Mutex lock can only be unlocked by the thread which lock it.(One of the main differences between mutex_lock and semaphore)

Mutex Variable (2)

* Header file:
  * #include <pthread.h>
* Declaration:
  * Mutex variable: pthread_mutex_t myMutex;
* Routines:
  * pthread_mutex_init(&myMutex, myAttr);
  * pthread_mutex_destroy(&myMutex);
  * pthread_mutex_lock(&myMutex);
  * pthread_mutex_unlock(&myMutex);
  * pthread_mutex_trylock(&myMutex);

Mutex Variable (3)

* 1. Create a mutex
  * Dynamically:
    * pthread_mutex_init(&myMutex, &myAttr);
  * Statically:
    * pthread_mutex_t myMutex = **PTHREAD_MUTEX_INITIALIZER**;
  * Return values: success – **0**; otherwise – **errno**
  * Use pthread_mutex_destroy(&myMutex) to free a mutex.

Mutex Variable (4)

* 1. Lock & unlock a mutex
  * A mutex is initialized as **unlocked**.
  * Function calls:
    * pthread_mutex_lock(&myMutex);
    * pthread_mutex_trylock(&myMutex);
    * pthread_mutex_unlock(&myMutex);
  * Return values: success : **zero**; otherwise : **errno**
  * **trylock** vs. **lock**:
    * When the mutex is **unlocked**: same to each other
    * When the mutex is already **locked**: return immediately vs. suspend

Mutex Variable (5)

* pthread_mutex_trylock(&myMutex);
* return immediately no matter mutex is locked or not.

* Two cases:
  * 1. When mutex is in unlocked status, return 0.
  * 2. When mutex is locked by other thread already, return **EBUSY.**

pthread_mutex_trylock()

Allow your program to work in unblock way.

Sample Codes

* Example-1: **pt_mutex.c**
  * Multiple threads modify a global variable (critical section)

  * **$ gcc –pthread pt_mutex.c –o pmutex**
  * **$ ./pmutex**

* When we should use a mutex lock?

Condition Variable (1)

* What is a convar?
  * A condition variable is basically a container of threads that are waiting for a certain condition.
  * Main routines:
  * **int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t*mutex);**
  * **int pthread_cond_signal(pthread_cond_t *cond);**
  * **pthread_cond_broadcast(pthread_cond_t *cond);**

Condition Variable (2)

* Header file:
  * #include <pthread.h>
* Declaration:
  * Condition variable: pthread_cond_t myConvar;
* Routines:
  * pthread_cond_init(&myConvar, NULL);
  * pthread_cond_destroy(&myConvar);
  * **pthread_cond_wait(&myConvar, &myMutex);**
  * **pthread_cond_signal(&myConvar);**
  * **pthread_cond_broadcast(&myConvar);**

Condition Variable (3)

* 1. Create a convar
  * Dynamically:
    * pthread_cond_init(&myConvar,&myAttr);
    * **myAttr** is usually ignored and specified as **NULL**.
  * Statically:
    * pthread_cond_t myConvar = **PTHREAD_COND_INITIALIZER**;
  * Return values: success – **0**; otherwise – **errno**
  * Use pthread_cond_destroy(&myConvar) to
    * free myConvar
    * or test if there are threads waiting myConvar

Condition Variable (4)

* 2. Wait a convar
  * Function calls:
    * **int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t*mutex);**
  * By calling pthread_cond_wait(&myConvar, &myMutex):
    * Automatically unlocks myMutex, and waits for myConvar to be signaled;
    * Calling thread is suspended ;
    * After myConvar is signaled, pthread is woke up, myMutex is relocked by thread again (automatically). Function return, need manually unlock.
  * Return values: success – **0**; otherwise – **errno**

Condition Variable (5)

* 3. Signal a convar
  * Function calls:

    int pthread_cond_signal(pthread_cond_t *cond);

    int pthread_cond_broadcast(pthread_cond_t *cond);
  * By calling pthread_cond_signal(&myConvar):
    * Wake up one of the threads that are waiting for myConvar;
  * By calling pthread_cond_broadcast(&myConvar):
    * Wakes up all threads that are currently waiting on myConvar;
    * Only **one** thread can get myMutex, and the others will be suspended.
  * Return values: success – **0**; otherwise – **errno**
  * Which **one** will get mutex?

Sample Codes

* Example-2: **pt_convar.c**
  * **$ gcc –pthread pt_convar.c –o pconvar**
  * **$ ./pconvar**

More about Convar

* Lose wakeup signal Pitfalls :
    - When there is no thread in the waiting list of convar, the wakeup signal sent by pthread_cond_signal(&convar) will disappear.

More about Mutex & Convar

* Man Page.
* Online Tutorials:
  * https://computing.llnl.gov/tutorials/pthreads/#Mutexes
  * http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThre_ads.html
  * http://randu.org/tutorials/threads/