CS 153 Lab6

Kishore Kumar Pusukuri
Outline

Mutex vs Condition Variables
Semaphores
Unlocking and locking mutex leads spinning or polling, wastes CPU time.

- We could sleep for some amount of time, but we do not know how long to sleep.
- A mutex is for locking and a condition variable is for waiting.
- Study the program prodcons_1.c
#include <pthread.h>

int pthread_cond_wait(pthread_cond_t *cptr, pthread_mutex_t *mptr);
int pthread_cond_signal(pthread_cond_t *cptr);

Both return: 0 if OK, positive Exxx value on error.

A conditional variable is a variable of type pthread_cond_t.
A mutex is always associated with a condition variable.

When we call `pthread_cond_wait` to wait for some condition to be true, we specify the address of the condition variable and the address of the associated mutex.

The term “signal” in the second function’s name does NOT refer to a SIGxxx signal.

Normally, `pthread_cond_signal` awakes one thread that is waiting on the condition variable.

Study the program `prodcons_2.c`
Mutexes are used to protect critical regions of code, so that only one thread at a time is executing within the critical region.

Sometimes a thread obtains a mutex lock and then discovers that it needs to wait for some condition to be true. When this happens, the thread waits on a condition variable.

A condition variable is always associated with a mutex.

The `pthread_cond_wait` function that puts the thread to sleep unlocks the mutex before putting the thread to sleep and relocks the mutex before waking up the thread at some later time.

The condition variable is signaled by some other thread, and that signaling thread has the option of waking up one thread using `pthread_cond_signal` or all the threads that are waiting for the condition to be true (using `pthread_cond_broadcast`).
 Mutex vs Condition Variables

Semaphores

Why semaphores?

- Mutexes and conditional variables can always be used to synchronize the various threads within a process.

- Posix also allows a mutex or conditional variable to be used for synchronization between multiple processes, “if the mutex and conditional variable is stored in memory that is shared between processes”.

- A semaphore is a primitive used to provide synchronization between various processes or between the various threads in a given process.

Types:

- Posix named semaphores.
- Posix memory-based semaphores.
- System V Semaphores.
Mutex vs Condition Variables
Semaphores

Posix named semaphores

Consider binary semaphores:

- Posix semaphores are identified by names that might correspond to pathnames in the file system.
- Three operations that a process can perform on a semaphore:
  - Create a semaphore.
  - Wait for a semaphore.
  - Post a semaphore.
wait operation:
while (semaphore_value <= 0)
    ; /* wait; i.e., block the thread or process */
semaphore_value--; /* we have the semaphore */

post/signal operation:
semaphore_value++;

Comparision of mutex and binary semaphore
to solve mutual exclusion problem:
initialize mutex; initialize semaphore sem to 1;
pthread_mutex_lock(&mutex) sem_wait(&sem)
critical region critical region
pthread_mutex_unlock(&mutex) sem_post(&sem);
Semaphore functions:
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag,...
               /*mode_t mode, unsigned int value */ );

Somewhat similar to syntax of file functions.
int sem_close(sem_t *sem);
int sem_wait (sem_t *sem);
int sem_post(sem_t *sem);
..and several other operations....

The above return: 0 if OK, -1 on error.
How to attack Project 2?
Mutex vs Condition Variables
Semaphores

Thaaaank
Yooooouuu!!!