Mutexes, Condition Variables, and Semaphores, Oh My!!!

Tom Kelliher, CS 411

1 Announcements

2 Discussion

2.1 Mutexes and Condition Variables

Used together.

Mutexes. Can be used alone.

```
#include <pthread.h>

pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_lock(&m);

// Critical section

pthread_mutex_unlock(&m);
```

Must include `pthread` library when compiling.

Condition Variables. Always used with an associated mutex.

```
#include <pthread.h>

pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;

pthread_mutex_lock(&m);
```

```
// Critical section

// Have to wait for condition to become true before
// continuing in CS.
//
// This thread had better be holding m.

while (<some condition>)
    pthread_cond_wait(&c, &m);

// Continued CS

// For the next two, assume this thread has done something to
// change the conditon associated with c.

// Signal one thread waiting on c to re-check its condition.
pthread_cond_signal(&c);

// Signal all threads waiting on c to re-check their conditon.
p_thread_cond_broadcast(&c);

pthread_mutex_unlock(&m);
```

Again, must include `pthread` library when compiling.

2.2 Semaphores

An alternate synchronization mechanism.

The textbook's code uses "zemaphores" rather than semaphores for the Apple platform. I don't know why...

Semaphores come in two "flavors":

- Binary
- Counting

```
#include <semaphore.h>

sem_t sem;
```

```
// Binary semaphore: Replace v with 0 or 1, depending upon the
// situation.
//
// Counting semaphore: Replace v with the initial count.  Again,
// situation-dependent.
//
// The second parameter is pshared.  For us, it will always be 0.
// See the documentation for other values.
sem_init(&sem, 0, v);

// Decrement the semaphore
// sem_wait(&sem);

// Increment the semaphore
// sem_post(&sem);
```

Again, must include `pthread` library when compiling.

The zemaphore code shows that semaphores can be implemented using mutexes and condition variables. Assuming it's possible to implement mutexes and condition variables using semaphores, these two synchronization mechanisms are equivalent.

3 Assignment

Run

```
git pull upstream main
```

to pull the "latest and greatest" additions to upstream.

In the `Concurrency-Sems` directory, study `pc.c`.

- Compile the program as is, `make pc`, and run it with these parameters: `./pc 10 1000 3`.
- Does the program run as expected?
- Find the three sections in the code where it says to swap the order of the calls to `sem_wait()`.
- Discuss whether or not swapping the calls in these three sections will have any effect upon the execution of the program.

- Make the swaps, re-compile the program, and run it again.
- Discuss any differences you find between what you expected and what happened, and why any differences occurred.