CS 220: Introduction to Parallel Computing

Review: Mutexes, Condition Variables

Lecture 25

Mutex Declaration

- **Where** you declare your mutex is very important
- For example, what happens when each thread creates its own mutex?
  - This is basically like checking if you have the keys to your own house
- In general, mutexes should be a **shared** resource
  - Declared globally

Mutex: Mental Model

- You can think of a mutex as a protector of a shared resource that only one thread can access at a time
- It’s the gatekeeper for your protected resource
- You’ll almost always have:
    1. The mutex
    2. The variable you’re protecting

Mutex: Mental Model

- Let’s say our shared resource is the whiteboard
- Before you can write on the whiteboard, you have to ask the instructor first
- The instructor will only allow one student to write on the board at a time
  - …if you request to use the whiteboard while someone else is already using it, then the instructor makes you wait

Checking a Mutex

- Thus far, we've just **locked** or **unlocked** a mutex
- What happens when we try to lock a mutex that is already locked by another thread?
  - We block!
- In some cases, we want to determine whether we can lock the mutex, but move on if we cannot:
  - pthread_mutex_trylock(&mutex)
  - Even if the mutex is already locked by another thread, the function call returns immediately

Back to the Whiteboard Example

- Now, assume that I've divided the whiteboard up into four quadrants
- I can now let four students have their own part of the whiteboard at a time
- To protect the four quadrants, we could have four unique mutexes
  - This doesn't scale very well… What happens when I buy another whiteboard or divide it up more?

Array of Mutexes

- One approach would be to keep a big array of mutexes, one for each part of the whiteboard
- Do we really need all that complexity?
- There is, however, a better way: **condition variables**

Condition Variables

- To wait for something to happen, we can use **condition variables**
- Condition variables have two related functions:
  - wait – wait for the condition to become true
  - signal – inform the waiting thread that the condition has changed
- When a thread is waiting, it **blocks**
  - Just like how our MPI programs block when they are waiting for a message to come in

Initializing Condition Variables

- Initialization is just like a mutex:

```
pthread_cond_t cond_variable =
    PTHREAD_COND_INITIALIZER;
```

- Note: to use a condition variable, you also need a mutex
  - Why? This protects the condition variable logic

Using Condition Variables

Thread A:

```
    pthread_mutex_lock(&mutex);
    while (num_students_at_board >= 4) {
        /* Note: mutex is released here: */
        pthread_cond_wait(&cond, &mutex);
    }
    /* Do the work we were waiting to do! */
    pthread_mutex_unlock(&mutex);
```

Thread B:

```
    pthread_mutex_lock(&mutex);
    /* Do whatever thread A is waiting for us to do ... */
    /* Signal the other thread! */
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
```