CS 220: Introduction to Parallel Computing

Condition Variables

Lecture 24

Remember: Creating a Thread

```
int pthread_create(
        pthread_t *thread,
        const pthread_attr_t *attr,
        void *(*start_routine)(void *),
        void *arg);
```

Passing Arguments to Threads

- pthread_create is very specific about what we can pass in
- In fact, we can only pass in a pointer
  - We've been using (abusing?) this to pass in our thread ID
- So how do we pass more than one argument to a new thread?
  - The answer: structs

Arg Struct Example

```
struct thread_params {
    char thread_name[50];
    unsigned int thread_id;
    unsigned long nonce;
}
struct thread_params *tp =
    malloc(sizeof(struct thread_params));
strcpy(tp->thread_name, "My Thread");
tp->thread_id = 10;
```

Passing it In

```
struct thread_params *tp =
    malloc(sizeof(struct thread_params));
strcpy(tp->thread_name, "My Thread");
tp->thread_id = 10;


pthread_create(
    &thread_handle,
    NULL,
    thread_func,
    tp);
```

Using the Struct

```
void *thread_func(void *input_ptr) {
    struct thread_params *tp
        = (struct thread_params *) input_ptr;
    printf("Starting Thread: %s\n",
        tp->thread_name);

    /* Your code here… */

}
```

Waiting for Changes (1/2)

- We discussed how busy waiting is one way to prevent access to a **critical section**
- Unfortunately, busy waiting is very inefficient!
- We have a better way: **mutexes**
- What about when we want to wait for something to happen before our thread does its work?
  - For example: I will wait until I receive a "go" message before I process this file

Waiting for Changes (2/2)

- We can busy wait on a variable to change
  - Once the change happens, we know we can proceed
  - Once again, this is inefficient
- Consider:
  - We have two threads, A and B
  - Thread A preprocesses the input file
  - Thread B calculates the statistics
  - In this case, thread B needs to wait for A

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

Blocking vs Waiting

- The big difference between blocking and actively waiting is efficiency
- Rather than constantly checking, go to sleep and let the operating system wake you up when something happens
  - Are we there yet?
    - Are we there yet?
      - Are we there yet?
        - Are we there yet?

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 (!condition) {
    /* 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);
```

Producer-Consumer

- We can use condition variables to implement **producer-consumer** synchronization
- Thread 1: **Producer** – creates the tasks
- Thread 2: **Consumer** – waits for tasks and carries them out
- This is a widely-used paradigm!
  - Work queues

Producer-Consumer: Example

Thread A: (Consumer)

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

Thread B: (Producer)

```
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);
```

pthreads: What we've Learned

- How to create a thread
- Busy waiting
- Mutexes
- Critical sections
- Condition variables