Mutexes and Condition Variables

Daniel Zappala

CS 360 Internet Programming
Brigham Young University

Mutexes

Mutex

- lock that allows only one thread into a critical section

```
1  #include <pthread.h>
2
3  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
4
5  int pthread_mutex_lock(pthread_mutex_t *mutex);
6  int pthread_mutex_trylock(pthread_mutex_t *mutex);
7  int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- must initialize the mutex first
- `pthread_mutex_lock()` will block if mutex is already locked
- `pthread_mutex_trylock()` will return EBUSY if mutex is locked

▸ GitHub

Don’t Use Busy Waiting!

Busy Waiting

```
1  while running {
2    c = NULL;
3    pthread_mutex_lock(&mutex);
4    if queue.not_empty() {
5      c = queue.dequeue();
6    }
7    pthread_mutex_unlock(&mutex);
8    if c {
9      /* handle connection */
10   }
11 }
```

- must busy wait until a connection is available
- wastes CPU time on a server that does not handle many connections

Condition Variables

Condition Variables

```
1  #include <pthread.h>
2  pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
3
4  int pthread_cond_wait(pthread_cond_t *cond,
5                        pthread_mutex_t *mutex);
6
7  int pthread_cond_signal(pthread_cond_t);
```

- must initialize the condition variable first
- `pthread_cond_wait()` will block until the condition is signaled; the thread now owns the mutex as well
- need a corresponding `pthread_cond_signal()` to wake up

Using Condition Variables

```
1  while running {
2    c = NULL;
3    pthread_mutex_lock(&mutex);
4    while queue.empty() {
5      pthread_cond_wait(&cond,&mutex);
6    }
7    c = queue.dequeue();
8    pthread_mutex_unlock(&mutex);
9    /* handle connection */
10 }
```

- process inserting into queue should signal condition when queue goes from empty to having at least one item
- **must re-check queue status when conditional wait returns**
- no guarantee that queue will be empty when you return

Timed Wait and Broadcast Signals

```
1  #include <pthread.h>
2
3  int pthread_cond_timedwait(pthread_cond_t *cond,
4                             pthread_mutex_t *mutex,
5                             const struct timespec *abstime);
6
7  int pthread_cond_broadcast(pthread_cond_t *cond);
```

- `pthread_cond_timedwait()` needs an absolute time; use `clock_gettime()` and add the length of time you want to wait
- `pthread_cond_broadcast()` wakes up all threads waiting for a signal

Producer Consumer

Producer Consumer Problem

- one or more producers are generating data and placing them in a buffer
- one or more consumers are taking items out of the buffer
- only one producer or consumer may access the buffer at any time

Producer Consumer

producer:

```
1  while (true) {
2    item = produce();
3    buffer.append(item);
4  }
```

consumer:

```
1  while (True) {
2    item = buffer.remove();
3    consume(item);
4  }
```

Producer Consumer

```
1  pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
2  pthread_cond_t not_empty = PTHREAD_COND_INITIALIZER;
3  pthread_cond_t not_full = PTHREAD_COND_INITIALIZER;
```

producer:

```
1  while (True) {
2    item = produce();
3    pthread_mutex_lock(&lock);
4    while buffer.full() {
5      pthread_cond_wait(&not_full,&lock);
6    }
7    buffer.append(item);
8    pthread_cond_signal(&not_empty);
9    pthread_mutex_unlock(&lock);
10 }
```

consumer:

```
1  while (True) {
2    pthread_mutex_lock(&lock);
3    while buffer.empty() {
4      pthread_cond_wait(&not_full,&lock);
5    }
6    item = buffer.remove();
7    pthread_cond_signal(&not_full);
8    pthread_mutex_unlock(&lock);
9    consume(item);
10 }
```

Looking at the Code ...

1. *What is the purpose of the mutex called lock?*
2. *What is the purpose of the condition variable called not_full?*
3. *What is the purpose of the condition variable called not_empty?*
4. *Why do we use a while() statement when waiting for the condition instead of an if() statement?*
5. *Always use signal while the process still holds the mutex.*