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

---

```c
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
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
Don’t Use Busy Waiting!
Busy Waiting

```c
while running {
    c = NULL;
    pthread_mutex_lock(&mutex);
    if queue.not_empty() {
        c = queue.dequeue();
    }
    pthread_mutex_unlock(&mutex);
    if c {
        /* handle connection */
    }
}
```

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

# Condition Variables

- 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

```c
#include <pthread.h>
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);

int pthread_cond_signal(pthread_cond_t);
```
Using Condition Variables

```c
while running {
    c = NULL;
    pthread_mutex_lock(&mutex);
    while queue.empty() {
        pthread_cond_wait(&cond,&mutex);
    }
    c = queue.dequeue();
    pthread_mutex_unlock(&mutex);
    /* handle connection */
}
```

- 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. int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);
3. 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:**

```java
while (true) {
    item = produce();
    buffer.append(item);
}
```

**consumer:**

```java
while (True) {
    item = buffer.remove();
    consume(item);
}
```
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:**

```c
while (True) {
    item = produce();
    pthread_mutex_lock(&lock);
    while buffer.full() {
        pthread_cond_wait(&not_full,&lock);
    }
    buffer.append(item);
    pthread_cond_signal(&not_empty);
    pthread_mutex_unlock(&lock);
}
```

**consumer:**

```c
while (True) {
    pthread_mutex_lock(&lock);
    while buffer.empty() {
        pthread_cond_wait(&not_empty,&lock);
    }
    item = buffer.remove();
    pthread_cond_signal(&not_full);
    pthread_mutex_unlock(&lock);
    consume(item);
}
```
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.