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 <thread>
2  #include <mutex>
3
4  mutex m;
5  m.lock();
6  // critical section
7  m.unlock();
```

- `m.lock()`
  - acquire the lock
  - will block if the mutex already locked
- `m.unlock()`
  - release the lock
- see example code in `mutex` directory

▸ GitHub

Don’t Use Busy Waiting!

Busy Waiting

```
1  while running {
2    c = NULL;
3    m.lock();
4    if queue.not_empty() {
5      c = queue.dequeue();
6    }
7    m.unlock();
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 <thread>
2  #include <mutex>
3  #include <condition_variable>
4
5  // shared data
6  mutex m;
7  condition_variable cv;
8  ready = false;
9
10 // thread 1
11 unique_lock<mutex> lock(m);   // acquires lock, unlocks when out of scope
12 ready = true;
13 cv.notify_one();
14
15 // thread 2
16 unique_lock<mutex> lock(m);  // acquires lock, unlocks when out of scope
17 while (not ready) {
18   cv.wait();
19 }
```

- `cv.wait()` will block until the condition is signaled, gives up the lock while waiting
- `cv.signal()` will wake up one thread waiting on signal, give it the lock

Condition Variables

```
1  #include <thread>
2  #include <mutex>
3  #include <condition_variable>
4
5  // shared data
6  mutex m;
7  condition_variable cv;
8  ready = false;
9
10 // thread 1
11 unique_lock<mutex> lock(m);     // acquires lock, unlocks when out of scope
12 ready = true;
13 cv.notify_one();
14
15 // thread 2
16 unique_lock<mutex> lock(m);     // acquires lock, unlocks when out of scope
17 while (not ready) {
18   cv.wait();
19 }
```

- signal will be lost if no thread is waiting for it!
- must loop to check the condition (ready==true) in case woken spuriously

Condition Variables

```
1  #include <thread>
2  #include <mutex>
3  #include <condition_variable>
4
5  // shared data
6  mutex m;
7  condition_variable cv;
8  ready = false;
9
10 // thread 1
11 unique_lock<mutex> lock(m);     // acquires lock, unlocks when out of scope
12 ready = true;
13 cv.notify_one();
14
15 // thread 2
16 unique_lock<mutex> lock(m);     // acquires lock, unlocks when out of scope
17 while (not ready) {
18   cv.wait(&m);
19 }
```

- see example code in `condition` directory

▸ GitHub

Timed Wait and Broadcast Signals

- use `cv.wait_for(lock,chrono::milliseconds(100))` to wait for a condition for at most 100 milliseconds
- use `cv.notify_all()` to wake up all threads waiting for a condition

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  mutex m;
2  condition_variable not_empty;
3  condition_variable not_full;
```

producer:

```
1  while (True) {
2    item = produce();
3    unique_lock<mutex> lock(m)
4    while buffer.full() {
5      not_full.wait(&m);
6    }
7    buffer.append(item);
8    not_empty.signal();
9    lock.unlock();
10 }
```

consumer:

```
1  while (True) {
2    unique_lock<mutex> lock(m)
3    while buffer.empty() {
4      not_empty.wait(&m);
5    }
6    item = buffer.remove();
7    not_full.signal();
8    lock.unlock();
9    consume(item);
10 }
```

Looking at the Code ...

1. *What is the purpose of the mutex called m?*
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.*