

Today

- Concurrency Problems
 - Producer Consumer
 - Bounded Buffer
 - Pipes
 - Dining Philosophers
- Synchronization Mechanisms
 - Condition Variables

Oct 28, 2015

Sprengle - CSC330

1

Review

- What is a condition variable?
 - What is its API?
- When do we use a condition variable?

Oct 28, 2015

Sprengle - CSC330

2

Review: Ping-Pong using a condition variable

```
void
PingPong() {
  mx.acquire();
  while(not done) {
    while(!myTurn)
      cv.wait(mx);
    do stuff;
    cv.signal();
  }
  mx.release();
}
```

```
wait (lock){
  release lock      Atomic
  put thread on wait queue
  go to sleep
  // after wake up
  acquire lock
}

signal (){
  wakeup one waiter (if any)
}
```

If blue calls cv.signal(),
why doesn't purple immediately run?

Oct 28, 2015

Sprengle - CSC330

3

Bounded-Buffer Problem

- Have a producer thread creating the items
- Have a consumer thread consuming the items
- Example: Soda machine
 - Producer adds a soda
 - Consumer removes a soda

```
consumer () {
  take a soda from machine
}
```

```
producer () {
  add one soda to machine
}
```

Oct 28, 2015

Sprengle - CSC330

4

Solving producer-consumer

- What are the variables/shared state?
 - Soda machine buffer
 - Number of sodas in machine ($\leq \text{MaxSodas}$)
- Locks?
 - 1 to protect all shared state (sodaLock)
- Mutual exclusion?
 - Only one thread can manipulate machine at a time
- Ordering constraints?
 - Consumer must wait if machine is empty (CV hasSoda)
 - Producer must wait if machine is full (CV hasRoom)

Oct 28, 2015

Sprengle - CSC330

5

Producer-consumer code

```
consumer () {
  lock
  wait if empty

  take a soda from machine

  notify (not full)
  unlock
}
```

```
producer () {
  lock
  wait if full

  add one soda to machine

  notify (not empty)
  unlock
}
```

Oct 28, 2015

Sprengle - CSC330

6

Producer-consumer code

```

consumer () {
    sodaLock.acquire()

    while (numSodas == 0) {
        hasSoda.wait(sodaLock)
    } CV1 IMx

    take a soda from machine

    hasRoom.signal() CV2
    sodaLock.release()
}

producer () {
    sodaLock.acquire()

    while (numSodas==MaxSodas){
        hasRoom.wait(sodaLock)
    } CV2 IMx

    add one soda to machine

    hasSoda.signal() CV1
    sodaLock.release()
}

```

Oct 28, 2015

Sprengle - CSC330

7

>1 Resource, >1 Consumers

The signal should be a *broadcast* if the producer can produce more than one resource, and there are multiple consumers.

```

consumer () {
    sodaLock.acquire()

    while (numSodas == 0) {
        hasSoda.wait(sodaLock)
    }

    take a soda from machine

    signal(hasRoom)

    sodaLock.release()
}

producer () {
    sodaLock.acquire()

    while (numSodas==MaxSodas){
        hasRoom.wait(sodaLock)
    }

    fill machine with soda

    broadcast(hasSoda)

    sodaLock.release()
}

```

Oct 28, 2015

Sprengle - CSC330

8

Broadcast vs signal

- Can I always use broadcast instead of signal?
 - Yes, assuming threads recheck condition
 - And they should: "loop before you leap"!
 - Another thread could get to the lock before wait returns
- Why might I use signal instead?
 - Efficiency -- May wakeup threads for no good reason

Oct 28, 2015

Sprengle - CSC330

9

Condition Variable Design Pattern

```

methodThatWaits() {
    lock.acquire();
    // Read/write shared
    // state

    while (
        testSharedState() {
            cv.wait(lock);
        }

    // Read/write shared
    // state
    lock.release();
}

methodThatSignals() {
    lock.acquire();
    // Read/write shared
    // state

    // If testSharedState is
    // now true
    cv.signal(lock);

    // Read/write shared
    // state
    lock.release();
}

```

Oct 28, 2015

Sprengle - CSC330

10

Summary: Condition Variables

- Condition variable is memoryless
 - If signal when no one is waiting, no op
- Wait *atomically* releases lock
 - What if wait, then release?
 - What if release, then wait?

```

wait (lock){
    release lock Atomic
    put thread on wait queue
    go to sleep
    // after wake up
    acquire lock
}

```

Oct 28, 2015

Sprengle - CSC330

11

Summary: Condition Variables

- When a thread is woken up from wait, it may not run immediately
 - Signal/broadcast puts thread on *ready* (not running) list
 - When lock is released, anyone might acquire it
- Benefit: simplifies implementation
 - Of condition variables and locks
 - Of code that uses condition variables and locks

Oct 28, 2015

Sprengle - CSC330

12

Using Condition Variables

- Document the condition(s) associated with each CV.
 - What are the waiters waiting for?
 - When can a waiter expect a signal?
- ALWAYS hold lock when calling wait, signal, broadcast
 - Condition variable is sync FOR shared state
 - ALWAYS hold lock when accessing shared state

Oct 28, 2015

Sprenkle - CSCI330

13

Using Condition Variables

- Wait MUST be in a loop -- “Loop before you leap!”

```
while (needToWait()) {  
    condition.wait(lock);  
}
```

 - Another thread may beat you to the mutex.
 - The signaler may be careless.
 - Some thread packages have “spurious wakeups”: 2 threads woken up, though a single signal has taken place
 - A single CV may have multiple conditions
 - Signals on CVs do not stack!
 - A signal will be lost if nobody is waiting: always check the wait condition before calling wait.

Oct 28, 2015

Sprenkle - CSCI330

14

Looking Ahead

- More synchronization mechanisms
 - Semaphores
- More synchronization problems
 - Dining Philosophers
- Still working through Chapter 5 of text book
- Project 3: next Wednesday

Oct 28, 2015

Sprenkle - CSCI330

15