9/8/2010

Locks ONLY solve Mutual Exclusion

They cannot solve sequencing problems

We another primitive- Monitors

# Monitors have 3 Components

1. Lock for mutual exclusion

2. One, or more, condition variables for sequencing

3. Monitor variables for making sequencing decisions
   - just data

shared data

# Condition Variables

Have no state

Have 3 operations

Wait: Put myself to sleep, waiting on some condition

Signal: Wake up 1 waiting thread - if there is one

Broadcast: Wake up all waiting threads

# Form of a Monitor

Acquire a lock - "Entering the monitor"

{

Access/manipulate our monitor variable(s) to make synchronization decisions

Use C.V. operations to wait, signal, or broadcast

}

Release the lock - "Exiting the monitor"

Example: Do I get in "Line"?

shared data:
```
int LineLength = 0; // monitor variable
Lock* LineLock = new Lock("—");
Condition* LineCV = new Condition("—");
```

```
// To see if I get in Line
LineLock->Acquire();
if( LineLength > 0 ) {
    LineCV->Wait( LineLock );
    LineLength--;
}
// Not in Line - ready to place an order
LineLock->Release();
```

LineLength++;

inside Condition  Lock* waitingLock = NULL;

```
void Condition::Wait(Lock* lock){
    disable interrupts

    if (lock == NULL){
        print msg, restore interrupts, return
    }

    if(waitingLock == NULL){
        // First thread calling Wait, Save the lock
        waitingLock = lock;
    }

    // Make sure the input lock matches
    // the saved lock
    if(waitingLock != lock){
        // Locks must match
        print msg, R.I. & return
    }
→   // Everything OK to be a "waiter"
    // Add thread to Condition wait Q
    lock->Release();
    currentThread->Sleep();
    lock->Acquire();
    restore interrupts
}
```

```
void Condition::Signal(Lock* lock){
    disable interrupts

    //If no waiters, restore interrupts & return
    if(waitingLock != lock){
        print msg, R.I. & return
    }

    //Wakeup 1 waiter
```

- Remove 1 thread from Condition Wait Q
- Put them on Ready Q (@ back)

```
    if(/*no more waiting threads*/){
    waitingLock = NULL;
    }

    R.I.
}
```

```
void Condition :: Broadcast (Lock* lock){
    .
    .
    while (/* there are waiters */){
        Signal (lock);
    }
    .
    .
}
```

Lock* lock1, lock2;

| Signaler | Waiter |
| --- | --- |
| lock1->Acquire(); | |
| lock2->Acquire(); | lock1->Acquire(); |
| cv1->Broadcast(lock2); | cv1->Wait(lock1); |

# A Bit More on Monitor Theory

Situation

Thread P is running in the CPU & the monitor

Thread Q, was in the monitor, but is now waiting on some condition

P executes a Signal to wakeup Q

Problem: P & Q cannot both be in the monitor @ the same time

Only Real Choice : Q has to wait for access to the monitor

↓

Mesa-style

Example: Producer / Consumer

"makes" an item

"consumes" an item

use a monitor

```
int itemCount = 0; // Number of produced items

// Assume an infinite buffer

Lock monitorLock;

Condition needItem; // For consumers to wait for an item
```

```
Producer() {
    while (true) {

        monitorLock.Acquire();
            //Produce an item
            //Put in buffer

        • itemCount ++;

        • needItem.Signal(&monitorLock);

        monitorLock.Release();
    }
}
```

```
Consumer(){
    while (true){
        monitorLock.Acquire();
        while( itemCount == 0){
            needItem.Wait(&monitorLock);
        }
        // There is @ least one item to be consumed
        //Consume 1 item
        itemCount--;
        monitorLock.Release();
    }
}
```

Mesa-style monitor

# Project 1 Part 2

A series of Producer/Consumer problems

Design/Code 1 interaction at a time

- what lock? name it
- what monitor variable(s)? name them
- what condition variable(s)?
- write pseudocode for design