CS 537
Lecture 22
Condition Variables/Monitors

Michael Swift

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Two Classes of Synchronization Problems

- Uniform resource usage with simple scheduling constraints
  - No other variables needed to express relationships
  - Use one semaphore for every constraint
  - Examples: thread join and producer/consumer
- Complex patterns of resource usage
  - Cannot capture relationships with only semaphores
  - Need extra state variables to record information
  - Use semaphores such that
    - One is for mutual exclusion around state variables
    - One for each class of waiting
- Always try to cast problems into first, easier type

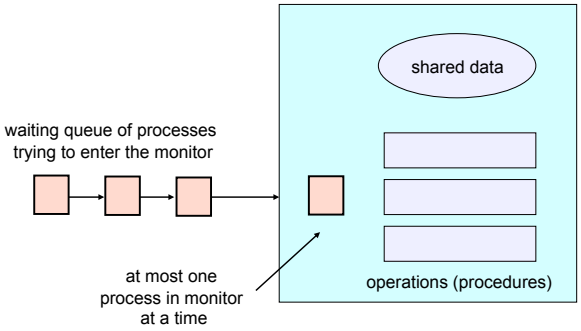
© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Monitors

- A programming language construct that supports controlled access to shared data
  - synchronization code added by compiler, enforced at runtime
  - why does this help?
- Monitor is a software module that encapsulates:
  - shared data structures
  - procedures that operate on the shared data
  - synchronization between concurrent processes that invoke those procedures
- Monitor protects the data from unstructured access
  - guarantees only access data through procedures, hence in legitimate ways

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

A monitor



© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Monitor facilities

- Mutual exclusion
  - only one process can be executing inside at any time
    - thus, synchronization implicitly associated with monitor
  - if a second process tries to enter a monitor procedure, it blocks until the first has left the monitor
    - more restrictive than semaphores!
    - but easier to use most of the time
- Once inside, a process may discover it can't continue, and may wish to sleep
  - or, allow some other waiting process to continue
  - condition variables provided within monitor
    - processes can wait or signal others to continue
    - condition variable can only be accessed from inside monitor

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Condition Variables

- A place to wait; sometimes called a rendezvous point
  - Always used with a monitor lock
  - No value (history) associated with condition variable
- Three operations on condition variables
  - wait(c)
    - release monitor lock, so somebody else can get in
    - wait for somebody else to signal condition
    - thus, condition variables have wait queues
  - signal(c)
    - wake up at most one waiting process/thread
    - if no waiting processes, signal is lost
    - this is different than semaphores: no history!
  - broadcast(c)
    - wake up all waiting processes/threads

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Use in a Program

- As a library (pthreads)

```
pthread_mutex_t mu;
 pthread_cond_t co;
 boolean ready;
 void foo( ) {
 pthread_mutex_lock(&mu);
 while(!ready)
   pthread_cond_wait(&co, &mu);
 …
 ready = TRUE;
 pthread_cond_signal(&co);  // unlock and signal
 pthread_mutex_unlock(&mu);
```

- As a language (Java)

```
synchronized withdraw(int amount) {
  while (balance < amount) {
    wait();
  balance -= amount;
  if (balance == 0) {
    notify();
}
```


© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Signaling

- signal(c) means
  - Wake one thread waiting on this condition variable (if any)
    - Signaller can keep lock and CPU
  - waiter is made ready, but the signaller continues
    - waiter runs when signaller leaves monitor (or waits)
    - condition is not necessarily true when waiter runs again
  - signaller need not restore invariant until it leaves the monitor
  - being woken up is only a hint that something has changed
    - must recheck conditional case
- Broadcast (or NotifyAll)
  - Wake all threads waiting on condition variable
  - Avoids need for multiple condition variables

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Producer/Consumer: pthread monitors

- Another thread may be scheduled and acquire lock before signalled thread runs
- Implication: Must recheck condition with while() loop instead of if()

Shared variables

```
cond_t not_empty, not_full;
int slots = 0;
mutex_t lock;
```

Producer

```
While (1) {
   mutex_lock(&lock);
   while (slots==N)
        cond_wait(&not_full,&lock);
   myi = get_empty(&buffer);
   Fill(&buffer[myi]);
   slots++;
   cond_signal(&not_empty);
   mutex_unlock(&lock);
}
```

Consumer

```
While (1) {
   mutex_lock(&lock);
   while (slots==0)
        cond_wait(&not_empty,&lock);
   myj = get_filled(&buffer);
   Use(&buffer[myj]);
   slots--;
   cond_signal(&not_full);
   mutex_unlock(&lock);
}
```


© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Traffic light

```
struct traffic_light{
   enum direction = {left, right};
   enum color = {green, yellow, red};
   color current_color[direction] = {green, red};
   cond_t changed[direction];
   direction current_dir = left;
   int in_intersection = 0;
   mutex_t *lock;

enter_left(dir)
   mutex_lock(lock)
   while ((current_dir != dir) && (current_color != green))
     cond_wait(changed[dir], loock);
   in_intersection++;
   mutex_unlock(lock);
   return;

exit(dir)
   mutex_lock(lock)
   in_intersection--;
   if (in_intersection == 0) && (current_color[dir] == red)
     broadcast(changed[other_dir(dir)]);
   mutex_unlock(lock);
```


© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Traffic light

```
struct traffic_light{
   enum direction = {left, right};
   enum color = {green, yellow, red};
   color current_color[direction] = {green, red};
   cond_t changed[direction];
   direction current_dir = left;
   int in_intersection = 0;
   mutex_t *lock;


timer()
   mutex_lock(lock);
   switch(current_color[direction]) {
     case green:
          current_color[current_dir] = yellow;
     case yellow:
          current_color[current_dir] = red;
          current_dir = other_dir(current_dir);
          current_color[current_dir] = green;
          if (in_intersection == 0) {
                  broadcast(changed[current_dir]);
   }
   mutex_unock(lock);
```


© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Examples

- Traffic light
  - Only one direction of traffic can flow at a time
- Try more at home from the book!
  - I will correct them if you would like

© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift

Readers and Writers Monitor Example

```
Monitor ReadersNWriters {
  int WaitingWriters, WaitingReaders,
        NReaders, NWriters;
  Condition CanRead, CanWrite;

Void BeginWrite()  {
   if(NWriters == 1 ||
      NReaders > 0) {
     ++WaitingWriters;
     wait(CanWrite);
     --WaitingWriters;
   }
   NWriters = 1;
}

Void EndWrite() {
  NWriters = 0;
  if(WaitingReaders)
    Signal(CanRead);
  else Signal(CanWrite);
}
```

```
Void BeginRead() {
  if(NWriters == 1 ||
     WaitingWriters > 0) {
    ++WaitingReaders;
    Wait(CanRead);
    --WaitingReaders;
  }
  ++NReaders;
  Signal(CanRead);
}

Void EndRead() {
  if(--NReaders == 0)
    Signal(CanWrite);
}
```


© 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift