Monitors

Operating Systems
ECE344

Ashvin Goel
ECE
University of Toronto
Synchronization Methods

- Monitors
- Semaphores
Monitors: Mutual Exclusion

- Monitors provide systematic solution for both mutual exclusion and synchronization

- Mutual exclusion: Two requirements
  - Any shared data is accessed within methods of data structure
  - Methods acquire lock at the beginning, release lock at the end

- Thus all shared data is accessed in critical section
- Now programmer can focus on synchronization

```c
Enqueue(queue) {
    lock(l);
    ...
    unlock(l);
}

Dequeue(queue) {
    lock(l);
    ...
    unlock(l);
}
```
Monitors: Synchronization

- Synchronization
  - Threads use one or more condition variables within methods
  - Condition variables allow a thread to 1) wait, or 2) wake up another waiting thread

- Condition variable abstraction
  - `wait(cv, lock)`:  
    - Wait on condition until another thread signals it
    - While thread waits, lock is released
    - When thread is signaled, it wakes up, reacquires lock, returns
  - `signal(cv, lock)`:  
    - Wakeup one thread waiting on condition
    - If a signal occurs before a wait, signal does nothing

```java
Enqueue(queue) {
  lock(l);
  wait/signal;
  unlock(l);
}
```
Variable Initialization Using Monitors

```c
char *V = NULL;
lock l = FALSE;
cv init_cv;

// called by Thread T1
Init() {
    lock(l);
    V = malloc(...);
    // signal that V
    // is non-NULL
    signal(init_cv, l);
    ...
    unlock(l);
}

// called by Thread T2
Read() {
    lock(l);
    // wait until V is non-NULL
    wait(init_cv, l);
    assert(V);
    // read *V
    ...
    unlock(l);
}
```

- Is this code correct?
  - Signal can be lost
Variable Initialization Using Monitors

- Wait/signal within lock
  - Would there be a problem if lock/unlock was not used above?

```c
char *V = NULL;
lock l = FALSE;
cv init_cv;

// called by Thread T1
Init() {  
  lock(l);
  V = malloc(...);
  // signal that V
  // is non-NULL
  signal(init_cv, l);
  ...
  unlock(l);
}

// called by Thread T2
Read() {  
  lock(l);
  if (V == NULL) {
    // wait until V is non-NULL
    wait(init_cv, l);
  }
  assert(V);
  // read *V
  ...
  unlock(l);
  ...
```
Global variables:
buf[n], in, out;
lock l = FALSE;
cv full;
cv empty;

void send(char elem) {
    lock(l);
    while ((in-out+n)%n == n - 1) {
        wait(full, l);
    } // full
    buf[in] = elem;
    in = (in + 1) % n;
    signal(empty, l);
    unlock(l);
}

char receive() {
    lock(l);
    while (in == out) {
        wait(empty, l);
    } // empty
    elem = buf[out];
    out = (out + 1) % n;
    signal(full, l);
    unlock(l);
    return elem;
}

- This code works – no races, deadlocks, spinning!

Why use “while”, instead of “if”?
Global variables:
buf[n], in, out;
lock l = FALSE;
wait_queue full;
wait_queue empty;

```c
char receive() {
    lock(l);
    while (in == out) {
        unlock(l);
        thread_sleep(empty);
    } // empty
    elem = buf[out];
    out = (out + 1) % n;
    thread_wakeup(empty);
    unlock(l);
    return elem;
}

void send(char elem) {
    lock(l);
    while ((in-out+n)%n == n - 1) {
        unlock(l);
        thread_sleep(full);
    } // full
    buf[in] = elem;
    in = (in + 1) % n;
    thread_wakeup(empty);
    unlock(l);
} // full
```
Summary

- Threads use synchronization to order their operations
  - A thread waits on a condition; when another thread makes the condition occur, it wakes up waiting thread
  - Ad hoc solutions lead to races, deadlocks, spinning …

- Monitors provide a systematic method for synchronization
  - All shared data is accessed within critical section
  - Synchronization is performed using condition variables
    - Condition variables provide wait and signal operations

- Discussed solutions for variable initialization, producer-consumer problems using monitors

- Next lecture, semaphores
Think Time

- Why are locks, by themselves, not sufficient for solving synchronization problems?
- Why is a lock passed to wait and signal in a monitor?
- What is the broadcast operation on a condition variable?
Think Time Answers

- Why are locks, by themselves, not sufficient for solving synchronization problems?
  - (lock, unlock) is used together by each thread, and in that order, to ensure mutual exclusion. Synchronization problems require a more general primitive: conditional sleep and wakeup. One thread performs the sleep, the other the wakeup.

- Why is a lock passed to wait and signal in a monitor?
  - Wait releases the lock before sleeping and then reacquires the lock after waking up. Signal does not use the lock, but it must only be called when the current thread holds the lock. So signal can test this invariant, and report a programming bug otherwise.
What is the broadcast operation on a condition variable?

- the signal operation of a condition variable wakes up one waiting thread. The broadcast operation wakes up all threads that are waiting on the condition.