Announcements

- **Program #1**
  - Scores posted (re-grade requests due in a week)

- **Program #2**
  - Due next Thursday (3/2/17)
Using Semaphores

- **critical section**
  
  repeat
  
  P(mutex);
  
  // critical section
  
  V(mutex);
  
  // non-critical section
  
  until false;

- **Require that Process 2 begin statement S2 after Process 1 has completed statement S1:**

  semaphore synch = 0;

  Process 1
  
  S1
  
  V(synch)

  Process 2
  
  P(synch)
  
  S2
Implementing semaphores

- Busy waiting implementations
- Instead of busy waiting, process can block itself
  - place process into queue associated with semaphore
  - state of process switched to waiting state
  - transfer control to CPU scheduler
  - process gets restarted when some other process executes a signal operations
Implementing Semaphores

- **Declaration**
  
  ```
  type semaphore = record
    value: integer = 1;
    L: FIFO list of process;
  end;
  ```

- **P(S):**
  
  ```
  S.value = S.value - 1
  if S.value < 0 then {
    add this process to S.L block;
  }
  ```

- **V(S):**
  
  ```
  S.value = S.value + 1
  if S.value <= 0 then {
    remove process P from S.L wakeup(P);
  }
  ```

*Can be neg, if so, indicates how many waiting*

*Bounded waiting!!*
Writers Have Priority

reader
repeat
  P(z);
  P(rsem);
P(x);
  readcount++;
  if (readcount == 1) then
    P(wsem);
  V(x);
  V(rsem);
V(z);
readunit;
P(x);
  readcount--; if readcount == 0 then
  V(rsem);
V(x);
forever

writer
repeat
  P(y);
  writecount++:
  if writecount == 1 then
    P(rsem);
  V(y);
writeunit
V(wsem);
P(y);
  writecount--;
  if (writecount == 0) then
    V(rsem);
V(y);
forever;
Notes on readers/writers with writers getting priority

Semaphores $x, y, z, wsem, rsem$ are initialized to 1

readers queue up on semaphore $z$; this way only a single reader queues on $rsem$. When a writer signals $rsem$, only a single reader is allowed through

```
P(z);
  P(rsem);
  P(x);
  readcount++;
  if (readcount==1) then
    P(wsem);

  V(x);
  V(rsem);
V(z);
```
Sample Synchronization Problem

- **Class Exercise:**
  - **CMSC 412 Midterm #1 (Spring 1998) Q#3**

- Solve a variation of the readers-writers problem, in which multiple writers can write at the same time. Specifically, there are readers and writers. Up to 5 reads at the same time are allowed, but only one write at the same time are allowed. A read and a write at the same time is not allowed. Provide a solution using semaphores with the following properties:
  - no busy waiting.
  - starvation-free (i.e. a continuous stream of readers does not starve writers, and vice versa) is desirable but not compulsory (but you will lose some points).
  - you cannot use process ids and you cannot have a separate semaphore for every process.
Students Work

- Reviewed examples of student work from last time
- Common theme:
  - Missed the need to keep track of how many processes are waiting