# Announcements

- Reading
  - Today
    - 8.1-8.3, 8.6 (6th Ed)
    - 7.1-7.3, 7.6 (8th Ed)

- Project #2 is due next Th at 5:00 PM (3/2/17)
- Midterm #1 is 3/9/17 in class

# Synchronization Problem From Last Class: Variables

- Binary semaphore mutex=1
- Counting semaphore reader = 0
- Binary semaphore writer = 0
- Shared int nReaders = 0
- Shared int wReaders = 0
- Shared int nWriters = 0
- Shared int wWriters = 0

- Writers execute this code:

```
while (1) {
        P(mutex);
        if (nReader + wReader + nWriter == 0)  {
                nWriter++;
                V(mutex);
        } else {
                wWriter++;
                V(mutex);
                P(writer);
        }
         // Write operation;
        P(mutex);
        NWriter = 0;
        If (wReaders > 0) {
                Temp = min(wReaders,5)
                for i = 1 to temp {
                        V(readers)
                        nReaders++;
                        wReaders--;
           }
        } else if (wWriters > 0) {
                wWriters--;
                nWriters++; V(writer);
          } V(mutex);
  }
```

- Readers execute this code:

```
while (1) {
          P(mutex)
          if (nWriters + wWriter == 0 & nReader < 5) {
                  nReaders++;
                 V(mutex);
          } else {
                 wReaders++;
                 V(mutex);
                 P(reader);
         }
         // Read operation;
         P(mutex);
         nReaders--;
         if (wWriters > 0 & nReaders == 0) {
                 wWriters--;
                 nWriters++;
                 V(writer);
         } else if (wReaders > 0 & wWriters == 0) {
                 nReaders++;
                 wReaders--;
                 V(reader);
         }
         V(mutex);
}
```

# Deadlocks

- System contains finite set of resources
  - memory space
  - printer
  - tape
  - file
  - access to non-reentrant code
- Process requests resource before using it, must release resource after use
- Process is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set

# Formal Deadlocks

- 4 *necessary* deadlock conditions:
  - Mutual exclusion - at least one resource must be held in a non-sharable mode, that is, only a single process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource is released
  - Hold and wait - There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently held by other processors

# Formal Deadlocks

- – No preemption: Resources cannot be preempted; a resource can be released only voluntarily by the process holding it, after that process has completed its task
- – Circular wait: There must exist a set {P0,...,Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource held by P2 etc.

- Note that these are not sufficient conditions

# Detecting Deadlock Algorithm

- Variables:

  n is the number of processes

  m is the number of resource types

  - Available - vector of length m indicating the number of available resources of each type
  - Work - vector of length m indicating the number of currently available resources of each type
  - Allocation - n by m matrix defining number of resources of each type currently allocated to each process
  - Request is an m x n matrix indicating the number of additional resources requested by each process
  - Finish is a vector of length n (processes) indicating if we are finished checking that process

# Detecting Deadlock

1. Work = Available;

   foreach i in n

      if any of Allocation[i,*] != 0 Finish[i] = false

      else Finish[i] = true;

2. Find an *i* such that Finish[i] = false and

      Request[I,*] <= Work[i,*] if no such i, go to 4

3. Work[i,*] += Allocation[i,*] ;

   Finish[i] = true;

   goto step 2

4. If Finish[i] = false for some i, system is in deadlock

**Note: this requires m x $n^2$ steps**

# Example

- Two resources $R_1$ & $R_2$
  - one instance of $R_1$ and two of $R_2$
- Three process A, B, C
- Initial State:
  - A has $R_2$, B has $R_1$ and C has $R_2$
  - B wants $R_2$ and C wants $R_1$

| | A | B | C |
|---|---|---|---|
| $R_1$ | 0 | 1 | 0 |
| $R_2$ | 1 | 0 | 1 |

Allocation

| | A | B | C |
|---|---|---|---|
| $R_1$ | 0 | 0 | 1 |
| $R_2$ | 0 | 1 | 0 |

Wants

| | |
|---|---|
| **$R_1$** | **0** |
| $R_2$ | 0 |

Work

| A | B | C |
|---|---|---|
| False | False | False |

Finish