Announcements

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

- Project #2 is due next Friday at 6:00 PM (11/7/11)
Sample Synchronization Problem

- **Class Exercise:**
  - **CMSC 412 Midterm #1 (Spring 1998) Q#3**
  - Solution posted at:
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 \{P_0, ..., P_n\} of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2 etc.

  - Note that these are not sufficient conditions
Detecting Deadlock

Work is a vector of length m (resources)
Finish is a vector of length n (processes)
- Allocation is an n x m matrix indicating the number of each resource type held by each process
- Request is an m x n matrix indicating the number of additional resources requested by each process

1. Work = Available;
   if Allocation[i] != 0 Finish[i] = false else Finish[i] = true;
2. Find an i such that Finish[i] = false and Request[i] <= Work if no such i, go to 4
3. Work += Allocation; 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
Recovery from deadlock

- Must free up resources by some means
- Process termination
  - kill all deadlocked processes
  - select one process and kill it
    - must re-run deadlock detection algorithm again to see if it is freed.
- Resource Preemption
  - select a process, resource and de-allocate it
  - rollback the process
    - needs to be reset the process to a safe state
    - this requires additional state
  - starvation
    - what prevents a process from never finishing?
Deadlock Prevention

- Ensure that one (or more) of the necessary conditions for deadlock do not hold
- Hold and wait
  - guarantee that when a process requests a resource, it does not hold any other resources
  - Each process could be allocated all needed resources before beginning execution
  - Alternately, process might only be allowed to wait for a new resource when it is not currently holding any resource
Deadlock Prevention

- **Mutual exclusion**
  - Sharable resources do not require mutually exclusive access and cannot be involved in a deadlock.

- **Circular wait**
  - Impose a total ordering on all resource types and make sure that each process claims all resources in increasing order of resource type enumeration.

- **No Premption**
  - Virtualize resources and permit them to be preempted. For example, CPU can be preempted.