Announcements

- **Midterm is Thursday (3/10/16)**
  - Covers up through this Th lecture

- **Project #2 is due Th at 5:00 PM**

- **Project #1 Re-grade request deadline is Wed 3/2/16 at 11:00 am**
Deadlock Avoidance

- Require additional information about how resources are to be requested - decide to approve or disapprove requests on the fly
- Assume that each process lets us know its maximum resource request
- Safe state:
  - system can allocate resources to each process (up to its maximum) in *some order* and still avoid a deadlock
  - A system is in a safe state if there exists a *safe sequence*
Safe Sequence

- Sequence of processes \(<P_1, \ldots, P_n>\) is a safe sequence if for each \(P_i\), the resources that \(P_i\) can request can be satisfied by the currently available resources plus the resources held by all \(P_j, j<i\).
- If the necessary resources are not immediately available, \(P_i\) can always wait until all \(P_j, j<i\) have completed.
Banker’s Algorithm

- Each process must declare the maximum number of instances of each resource type it may need
- Maximum can’t exceed resources available to system
- 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
  - Max - n by m matrix defining the maximum demand of each process
  - Allocation - n by m matrix defining number of resources of each type currently allocated to each process
  - Need: n by m matrix indicating remaining resource needs of each process
  - Work: a vector of length m (resources)
  - Finish: a vector of length n (processes)
Safe State Predicate

1. Work = Available; Finish[*] = false
2. Find an i such that Finish[i] = false and Need[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] = true for all i, system is in a safe state

Note this requires m x n^2 steps
Safe State Predicate - Example

Three resources: A, B, C (10, 5, 7 instances each)

Consider the snapshot of the system at this time

<table>
<thead>
<tr>
<th>Alloc</th>
<th>Max</th>
<th>Avail</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>P0</td>
<td>0 1 0</td>
<td>7 5 3</td>
<td>3 3 2</td>
</tr>
<tr>
<td>P1</td>
<td>2 0 0</td>
<td>3 2 2</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>3 0 2</td>
<td>9 0 2</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>2 1 1</td>
<td>2 2 2</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>0 0 2</td>
<td>4 3 3</td>
<td></td>
</tr>
</tbody>
</table>

System is in a safe state, since the sequence <P1, P3, P4, P2, P0> satisfy the safety criteria.
Resource Request Algorithm

(1) If \( \text{Request}_i \leq \text{Need}_i \) then goto 2
   - otherwise - the process has exceeded its maximum claim

(2) If \( \text{Request}_i \leq \text{Available} \) then goto 3
   - otherwise process must wait since resources are not available

(3) Check request by having the system pretend that it has allocated the resources by modifying the state as follows:
   - \( \text{Available} = \text{Available} - \text{Request}_i \)
   - \( \text{Allocation} = \text{Allocation} + \text{Request}_i \)
   - \( \text{Need}_i = \text{Need}_i - \text{Request}_i \)

   • Find out if resulting resource allocation state is safe, otherwise the request must wait.