Announcements

- Midterm is Thursday (3/9/17)
  - Covers up through this Th lecture

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

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)
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 n² 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></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>P0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</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 Request\(_i\) \(\leq\) Need\(_i\), then goto 2
   - otherwise - the process has exceeded its maximum claim

(2) If Request\(_i\) \(\leq\) 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:
   - Available = Available - Request\(_i\)
   - Allocation = Allocation + Request\(_i\)
   - Need\(_i\) = Need\(_i\) - Request\(_i\)
   - Find out if resulting resource allocation state is safe, otherwise the request must wait.

Managing Memory

- Main memory is big, but what if we run out
  - use virtual memory
  - keep part of memory on disk
    - bigger than main memory
    - slower than main memory

- Want to have several program in memory at once
  - keeps processor busy while one process waits for I/O
  - need to protect processes from each other
  - have several tasks running at once
    - compiler, editor, debugger
    - word processing, spreadsheet, drawing program

- Use virtual addresses
  - look like normal addresses
  - hardware translates them to physical addresses
Advantages of Virtual Addressing

- Can assign non-contiguous regions of physical memory to programs
- A program can only gain access to its mapped pages
- Can have more virtual pages than the size of physical memory
  - pages that are not in memory can be stored on disk
- Every program can start at (virtual) address 0