Announcements

- Homework 2 is due.
  - No new homework this week.
- Project 1:
  - Deadline tonight at midnight.
- Project 2 is now online.

- Midterm Exam
  - Friday 7/17
  - Covers everything through today’s lecture
Review

• Scheduling Metrics
  – Average wait time, CPU utilization, etc

• Scheduling Algorithms
  – First come, first serve
  – Shortest Job First / Shortest Time Remaining First
  – Round Robin
  – Priority
Mars Pathfinder

- July 4, 1997 landing on Martial surface
- Series of software glitches started a few days after landing
  - Eventually debugged and patched remotely from Earth

Note: information and images adapted from [mdwelsh, harvard]
VxWorks Operating System

• Real-time OS (Wind River Systems)
  – OS uses a priority scheduler
    • Important for real-time tasks
  – Concurrent threads communicate on information bus
    • Shared memory area guarded by a monitor
VxWorks Operating System

• Real-time OS (Wind River Systems)
  – OS uses a priority scheduler
    • Important for real-time tasks
  – Concurrent threads communicate on information bus
    • Shared memory area guarded by a monitor

- Obtain mutex; write data
- Wait for mutex to read data
VxWorks Operating System

• Real-time OS (Wind River Systems)
  – OS uses a priority scheduler
    • Important for real-time tasks
  – Concurrent threads communicate on information bus
    • Shared memory area guarded by a monitor

![Diagram showing mutexes and threads communicating via information bus]

- Weather Data Thread
- Communication Thread
- Information Bus Thread
- Mutex
- Free mutex;
- Wait for mutex to read data
VxWorks Operating System

- Real-time OS (Wind River Systems)
  - OS uses a priority scheduler
    - Important for real-time tasks
  - Concurrent threads communicate on information bus
    - Shared memory area guarded by a monitor

Information Bus (Monitor)

Weather Data Thread
Communication Thread
Information Bus Thread

Obtain mutex; read data

 Mutex
VxWorks Operating System

• Introduce priorities to these threads
VxWorks Operating System

• Introduce priorities to these threads

Obtain mutex; write data
Wait for mutex to read data

Weather Data Thread
Low priority

Communication Thread
Medium priority

Information Bus Thread
High priority
VxWorks Operating System

• Introduce priorities to these threads
  – OS timer interrupt
  – Scheduler runs higher priority thread

Information Bus (Monitor)

Mutex

Weather Data Thread
Low priority

Communication Thread
Medium priority
Interrupt!

Information Bus Thread
High priority

Obtain mutex; write data
Wait for mutex to read data
VxWorks Operating System

• Introduce priorities to these threads
  – OS timer interrupt
  – Scheduler runs higher priority thread
• Communication thread has higher priority than weather data thread
• But the high priority information bus thread is stuck waiting
  – This scenario is called priority inversion
Priority Inversion

• Problem description
  – A high priority thread is stuck waiting for a low priority thread to finish
  – In this case, the medium priority thread was holding up the low-priority thread

• How do we fix this?
Priority Inheritance

- Low priority thread wins the race for the lock.
Priority Inheritance

- Low priority thread wins the race for the lock.
- Low priority thread temporarily upgraded to high priority.
Priority Inheritance

- Low priority thread wins the race for the lock.
- Low priority thread temporarily upgraded to high priority.
Priority Inheritance

- Low priority thread wins the race for the lock.
- Low priority thread temporarily upgraded to high priority.
- After releasing mutex, down graded to low priority.

• Low priority thread wins the race for the lock.
• Low priority thread temporarily upgraded to high priority.
• After releasing mutex, down graded to low priority.
Priority Inheritance

• Solution approach: **Priority Inheritance**
  – Allow a thread to *inherit* the priority of any thread that is waiting for it
  – Blocked, higher priority threads “donate” their priority to lower priority threads
    • Donation only occurs as long as the higher priority thread is blocked – once this is no longer true, threads retain their original priority

• For our example
  – Medium thread will no longer block low priority thread
  – For the Mars pathfinder, they uploaded the fix remotely and everything was fine after that
    • Able to replay bugs in lab; reset real machine to known “good state” and upload fix – good lessons for developing large, real-time systems
Starvation and Deadlock

• Two types of resources:
  – Preemptible: can take away from a thread (e.g., CPU..)
  – Non-preemptible: can’t take away from a thread (e.g., mutex, lock..)

• Starvation:
  – A thread never makes progress because other threads are using a resource it needs

• Deadlock:
  – A circular resource dependency between multiple threads (note: threads is interchangeable with processes, jobs, tasks in this context)

• Starvation is not the same as deadlock
Deadlock Example

// two bank accounts protected by a
// monitor, with an associated mutex

Monitor Alice, Bob, Eve;

// transfer from Alice to Bob
Alice.mutex.acquire();
Alice.withdraw(amount);
Bob.mutex.acquire();
Bob.deposit(amount);
...Alice.mutex.release();

// transfer from Bob to Alice
Bob.mutex.acquire();
Bob.withdraw(amount);
Alice.mutex.acquire();
Alice.deposit(amount);
...Bob.mutex.release();

Thread 1

Thread 2

Thread 1 and Thread 2 are Deadlocked!
Deadlock

• Processes that acquire multiple resources are dependent on those resources
  – E.g., locks semaphores, monitors, etc.

• What if one process tries to allocate a resource that a second process holds, and vice-versa?
  – Neither can make progress!

• We call this situation **deadlock** and we’ll look at:
  – Definition and conditions necessary for deadlock
  – Representation of deadlock conditions
  – Approaches to dealing with deadlock
Deadlock Definition

• **Definition:**
  – Deadlock exists among a set of processes if every process is waiting for an event that can be caused only by another process in the set.

• **Deadlock is a problem that can arise:**
  – When processes compete for access to limited resources
  – When processes are incorrectly synchronized
Conditions for Deadlock

Deadlock can exist *if and only if* four conditions hold:

- **Mutual exclusion** – At least one resource must be held in a non-sharable mode (i.e., only one instance)
- **Hold and wait** – There must be one process holding one resource and waiting for another resource
- **No preemption** – Resources cannot be preempted (i.e., critical sections cannot be aborted externally)
- **Circular wait** – There must exist a set of processes \{P_1, P_2, P_3, ..., P_n\} such that \(P_1\) is waiting for a resource held by \(P_2\), \(P_2\) is waiting for \(P_3\), ..., and \(P_n\) for \(P_1\)
Resource Allocation Graph

• Deadlock can be described using a resource allocation graph (RAG)
• The RAG consists of sets of vertices \( P = \{P_1, P_2, P_3, \ldots, P_n\} \) of processes and \( R = \{R_1, R_2, R_3, \ldots, R_m\} \) resources
  – A directed edge from a process to a resource \( P_i \rightarrow R_j \), implies that \( P_i \) has requested \( R_j \)
  – A directed edge from a resource to a process \( R_i \rightarrow P_j \), implies that \( R_i \) has been acquired by \( P_j \)
  – Each resource has a fixed number of units
• If the graph has no cycles, deadlock cannot exist
• If the graph has a cycle, deadlock may exist
A cycle...deadlock?

RAG Example

A cycle...deadlock?
Dealing with Deadlock

There are four ways to deal with deadlock:

• Ignore it
  – Do you feel lucky?

• Prevention
  – Make it impossible for deadlock to happen

• Avoidance
  – Control allocation of resources

• Detection and recovery
  – Look for a cycle in dependencies
Deadlock Prevention

Prevent at least one of necessary conditions:

• **Mutual exclusion**
  – Make resources sharable

• **Hold and wait**
  – Process can’t hold one resource when requesting another
  – Process requests, releases all needed resources at once

• **Preemption**
  – OS can preempt resource (costly)

• **Circular wait**
  – Impose an ordering (numbering) on the resources and request them in order (popular technique)
Deadlock Avoidance

• How to avoid deadlock
  – Require processes to provide information in advance about what resources will be needed
  – System grants resources to avoid circularities (wait dependencies)

• Tough to do
  – Hard to determine all resource needs in advance
  – Good theoretical problem, not as practical
Banker’s Algorithm

The Banker’s Algorithm is the classic approach to deadlock avoidance for resources with multiple units

• Assign a **credit limit** to each customer (task)
  – Maximum credit claim must be stated in advance

• Reject any request that leads to a **dangerous state**
  – A dangerous state is one where a sudden request by any customer for the full credit limit could lead to deadlock
  – A recursive reduction procedure recognizes dangerous states

• In practice, the system must keep resource usage well below capacity to maintain a **resource surplus**
  – Rarely used in practice due to low resource utilization
## Banker’s Algorithm Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

**Available system resources**

**Currently allocated resources per Process**

**Processes (maximum resources)**
Banker’s Algorithm Example

A state is safe if it is possible for all processes to execute and terminate

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Available system resources

Currently allocated resources per Process

Processes (maximum resources)
### Banker’s Algorithm Example

A **state is safe** if it is possible for all processes to execute and terminate.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Available system resources**
- **Currently allocated resources per Process**
- **Processes (maximum resources)**

- **P2 can terminate**
A state is safe if it is possible for all processes to execute and terminate.
Banker’s Algorithm Example

A state is safe if it is possible for all processes to execute and terminate

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Available system resources

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Currently allocated resources per Process

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Processes (maximum resources)

P1 can terminate
Banker’s Algorithm Example

A state is safe if it is possible for all processes to execute and terminate

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Available system resources

Currently allocated resources per Process

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Processes (maximum resources)
Banker’s Algorithm Example

A state is safe if it is possible for all processes to execute and terminate.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available system resources</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currently allocated resources per Process</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>P3 can terminate</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Processes (maximum resources)</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Banker’s Algorithm Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Available system resources

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Currently allocated resources per Process

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Processes (maximum resources)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
**Banker’s Algorithm Example**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Available system resources**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Currently allocated resources per Process**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Processes (maximum resources)**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

P1’s request

P1’s request cannot be granted

P1 cannot terminate
### Banker’s Algorithm Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Available system resources:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Currently allocated resources:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

P1’s request:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

P2 cannot terminate.

Processes (maximum resources):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Banker’s Algorithm Example

Resource request is denied or delayed because it does not lead to a safe state.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Available system resources

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>

Currently allocated resources per Process

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Processes (maximum resources)

P3 cannot terminate
Deadlock Detection and Recovery

• Detection and Recovery
  – If we don’t have deadlock prevention or avoidance, then deadlock may occur
  – In this case, we need to detect deadlock and recover

• To do this, we need two algorithms
  – One to determine whether a deadlock has occurred
  – Another to recover from the deadlock

• Possible, but expensive (time consuming)
  – Implemented in VMS
  – Run detection algorithm if resource request times out
Deadlock Detection

• Detection
  – Traverse the resource graph looking for cycles
  – If a cycle is found, need to try and determine which resource or process to preempt

• Expensive
  – Many processes and resources to traverse

• Only invoke detection algorithm depending on
  – How often or likely deadlock is
  – How many processes are likely to be affected when it occurs
    • There is a tradeoff between overhead from running algorithm frequently, and being able to identify offending process
Deadlock Recovery

Once a deadlock is detected, we have two options...

• **Abort processes**
  – Abort all deadlocked processes
    • Processes need to start over again
  – Abort one process at a time until cycle is eliminated
    • System needs to rerun detection after each abort

• **Preempt resources** (force their release)
  – Need to select process and resource to preempt
  – Need to rollback process to previous state
  – Need to prevent starvation
Deadlock Summary

• Deadlock occurs when processes are waiting on each other and cannot make progress
  – Cycles in Resource Allocation Graph (RAG)

• Deadlock requires four conditions:
  – Mutual exclusion, Hold and wait, No resource preemption, circular wait

• Four approaches to dealing with deadlock:
  – Ignore it: risky
  – Prevention: make one of the four conditions impossible
  – Avoidance: banker’s algorithm (control allocation)
  – Detection and Recovery: look for a cycle, preempt or abort
Next Time

• Midterm Review
  – Bring questions
• Peerwise questions due tomorrow at midnight.
• Check Web site for course announcements
  – http://www.cs.ucsd.edu/classes/su09/cse120