Exam Review

Synchronization wrap-up

Deadlock Avoidance: Banker’s algorithm

Today

Ways of handling deadlock

- Deadlock avoidance
- Deadlock prevention
- Deadlock detection and recovery

- Circular wait
- No preemption
- Hold and wait
- Mutual exclusion

Necessary conditions for deadlock:

Last Class: Deadlocks
Otherwise, the thread must wait.

- Leaves the system in a safe state.
- The algorithm allocates resources to a requesting thread if the allocation may need for the duration of the execution.
- Force threads to provide advance information about what resources they force threads to provide advance information about what resources they require.
- This algorithm handles multiple instances of the same resource.

**Banker’s Algorithm**

**Deadlock Avoidance**

- This solution does not work for multiple instances of the same resource.
- The claim edge is converted to a request edge and the thread waits.
- The algorithm would result in an unsafe state, the allocation is denied even if the environment is available.
- A cycle in this extended resource allocation graph indicates an unsafe state.
- Changing the direction of a claim edge results in converting a claim edge to an allocation edge and vice versa.
- Claim edges: an edge from a thread to a resource that may be requested in the future.
```java
{ 
  if (need[i] + alloc[i] < max[i])
    alloc[i] = need[i] + alloc[i]; // allocate
  else // allocate - request:
    alloc[i] = need[i]; // see if the request could be satisfied
  if (alloc[i] > need[i])
    alloc[i] = need[i]; // allocate additional resources to satisfy the request

  if (request > need[i])
    request = need[i]; // request cannot be reduced below requested

  public void snaph巡onofAllocate (int resource, int i)
  }
}
```

---

**Banker's Algorithm: Resource Allocation**

---

**Preventing Deadlock with Banker's Algorithm**

---

```java
class ResourceManager{
  int n; // # of resources that each thread might still request
  int m; // # of each resource that each thread is using
  int[] max; // # of each resource that each thread may want
  int[] need; // # of available resources of each type
  int[] alloc; // # of resources
  int[] free; // # of threads
}
```
### Example using Banker's Algorithm

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>I</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>I</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

**System snapshot**

```java
private boolean bankersAlgorithm() {
    boolean finish = false;
    while (finish) {
        if (isAvailable()) {
            finish = true;
            int[] work = allocate(work);
        } else {
            int[] work = allocate(work + vector_operations);
            finish = true;
        }
    }
    return finish;
}
```

**Worst case: requires O(mn^3) operations to determine if the system is safe.**
What is a sequence of process execution that satisfies the safety constraint?

<table>
<thead>
<tr>
<th>Process</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>7</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

What would be the new system state after the allocation?

Algorithm: Grant the request immediately?

If a request from process P1 arrives for additional resources of (0, 5, 2), can the banker's algorithm grant the request immediately?

Example (cont'd)
and the sequence P₀, P₂, P₄ satisfies the safety constraint.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P₂</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P₄</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Example: Solutions

3. The new system state after the allocation is:

- P₀: 0, 1, 0
- P₂: 1, 0, 0
- P₄: 0, 0, 0

Since the new system state after the allocation is

\[ (0, 1, 0) + (1, 0, 0) = (1, 1, 0) \]

Yes, since the request can be granted immediately. Show the system state, and other criteria.

If a request from process P₀ arrives for additional resources of (0, 0, 0), can the bank's allocation grant the request immediately? Show the system state, and other criteria.

### Example: Solutions

Process asks for its maximum number of resources when it executes.

Yes, because the process can be executed in the sequence P₀, P₂, P₄, P₃, even if each is the system in a safe state? Why?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</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>P₄</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Need = Max - Allocation.

What is the content of the need matrix?

Resources = total + available

How many resources of type (A, B, C) (3, 0, 0)?

### Example: Solutions
What is the relationship between semaphores and locks?

- A counting semaphore enables simultaneous access to a fixed number of resources
  - Signal: unblocks a process on the wait queue; otherwise, increments value
  - Wait: Decrements value; Thread continues if value > 0 (semaphore is available)
  - Value: Initialization depends on problem

**Semaphores:**
- Else: the lock becomes free.
- Request: Enables another thread to get lock. If threads are waiting, one gets the lock.
- Acquire: Guarantees only one thread has lock. If another thread holds the lock, the
  - Value: Initially lock is always free.

**Locks:**
- High-Level Synchronization Primitives

What can the OS do with these low-level primitives?

<table>
<thead>
<tr>
<th>Testset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
</tr>
<tr>
<td>Interrupt</td>
</tr>
<tr>
<td>Load/Store</td>
</tr>
</tbody>
</table>

Advantages
- Low-Level Synchronization Primitives: hardware support

Synchronization Wrap up
Deadlocks

Ways of handling deadlock

- Circular wait
- No preemption
- Hold and wait
- Mutual exclusion

• Necessary conditions for deadlock:

Deadlocks

•

Operations.

Rule: thread must hold the lock when doing condition variable

- Broadcast() - wake up all waiting threads
- Signal() - wake up waiting thread (if one exists) and give it the lock
- Wait() - atomically release lock, go to sleep

Critical section Operations: A Condition Variable is a queue of threads waiting for something inside a

- Lock is initially free.
- Always release lock when finished with shared data.
- Always acquire lock before accessing shared data structure.
- Lock::Release - unlock, and wake up any thread waiting in Acquire.
- Lock::Acquire - wait until lock is free, then signal it.

Monitor locks provide mutual exclusion to shared data.

High-Level Synchronization Primitives: Monitors
7. What is the difference between a kernel thread and a user-level thread?
6. What is the difference between a process and a thread?
5. What is a context switch? What happens during a context switch? What causes a context switch to occur?
4. How does the OS keep track of processes?
3. What execution states can a process be in? What do they mean? What causes a process to change execution state?
2. What is a process control block? What is it used for? What information does it contain?
1. What is a process?

Topics you should understand:

Processes and Threads

Exam Review
Expect that variation to have.

2. Given a variation to a scheduling algorithm we studied, discuss what impact you would

time for each scheduling algorithm we have discussed.
the total CPU time you should be able to compute their completion time and waiting

I. Given a list of processes, their arrival time, the lengths of their CPU and I/O bursts, and

Things you should be able to do:

CPU Scheduling

Topics you should understand:

CPU Scheduling
and does it avoid deadlock?

Guarantee mutual exclusion when appropriate, does it avoid starvation,
able to explain whether you believe it works. In particular, does it

I. Given some code that uses locks, semaphores, or monitors, you should be

Things you should be able to do:

**Synchronization**

Impementation of critical sections? What are the advantages and disadvantages?

9. What is a test-and-set? How can a test-and-set instruction be used to support the

8. How can interrupts be manipulated to support the implementation of critical sections?

7. What is busy waiting?

disadvantages of each?

semantics after a condition variable has been signaled? What are the advantages and

6. What is a monitor? What is a condition variable? What are the two possible resemption

5. What is a semaphore? What are the three things a semaphore can be used for?

4. What is a lock? What do you need to do to use a lock correctly?

3. What is a critical section?

2. What is mutual exclusion?

1. Why do we need to synchronize processes/threads?

Topics you should understand:

**Synchronization**
prevent deadlock.

5. Given some code that might deadlock, describe how you might change the algorithm to
from a process, determine if the request can be safely satisfied.
requirements of processes, and available resources, and request for additional resources.
resources, and available resources, determine if the processes are deadlock.
resources, and available resources, determine if the state could lead to
resources, and available resources, determine if the processes are deadlock.

4. Given a state consisting of resources allocated to processes, maximum resource

3. Given a state consisting of resources allocated to processes, maximum resource

2. Given a state consisting of resources allocated to processes, maximum resource

1. Given some code, reason about whether or not it is possible for deadlock to occur.

Things you should be able to do:

Deadlocks

State?

4. What is a safe state? What is the difference between an unsafe state and a deadlocked

3. After detecting deadlock, what options are conceivable for recovering from deadlock?

2. What is the difference between deadlock detection and deadlock prevention?

1. What are the four necessary conditions for deadlock to occur?

Topics you should understand:

Deadlocks
...system, such as Unix, Mach, Windows NT, ...

- You will not be asked detailed questions about any specific operating system.
- You will be asked to write pseudo code with synchronization.
- You should be able to read C++ code.