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.

- Leave 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 need for the duration of the execution.
- This algorithm handles multiple instances of the same resource.

**Banker’s Algorithm**

- 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.
  - If the allocation would result in an unsafe state, the allocation is denied even if the resource is available.
  - A cycle in this extended resource allocation graph indicates an unsafe state.
  - Changing the direction of an edge that is in a cycle results in converting a claim edge to an allocation edge and
  - Claim edges: an edge from a thread to a resource that may be requested in the future.

**Deadlock Avoidance**
{ }
    // read the changes to avail, alloc, and need
    mat = ()
    // read the changes to alloc, need, and need;
    if (this is an unsharable, undo the allocation and wait
    )}

    // resource request
    need[i][t] = need[i][t] - resource;
    alloc[i][t] = alloc[i][t] + resource;
    avail = avail - resource; // vector address
    if (avail < resource)
    // see if the request could be on an unsharable state
    // enough resources exist to satisfy the request
    // insufficient resources available
    // wait();
    if (request < need[i][t])
    // the thread making the request
    // is the thread making the request
    // request contains the resources being requested
    public void synchronizeAlloc(int[] need, int[] alloc)
    // class ResourceManager
    }

    // Preventing Deadlock with Banker's Algorithm
    #ifndef "need[i][t]"
    #define "need[i][t]"
    #endif
    #ifndef "alloc[i][t]"
    #define "alloc[i][t]"
    #endif
    #ifndef "max[i][t]"
    #define "max[i][t]"
    #endif
    
    #ifndef "resources"
    #define "resources"
    #endif
    
    int i, n, m;
    int[] threads;
System snapshot:

Example using Banker's Algorithm

Worst case requires \( O(mn^3) \) operations to determine if the system is safe.

```java
    if (滿足[t] == true for all t)
        {return true;
            else
                return false;
        }
    [t] = true;
    [t] = [t] + [t];
    and needed [t] = [t];
    while (find a process that can complete the work now
        {find a process that can complete the work now
            work = [t] + [t];
            if (工作 = [t])
                work = [t] + [t];
                if (work = [t])
                    work = [t] + [t];
                    if (work = [t])
                        work = [t] + [t];
                        if (work = [t])
                            work = [t] + [t];
                            if (work = [t])
                                work = [t] + [t];
                                if (work = [t])
                                    work = [t] + [t];
                                    if (work = [t])
                                        work = [t] + [t];
                                        if (work = [t])
                                            work = [t] + [t];
                                            if (work = [t])
                                                work = [t] + [t];
                                                if (work = [t])
                                                    work = [t] + [t];
                                                    if (work = [t])
                                                        work = [t] + [t];
                                                        if (work = [t])
                                                            work = [t] + [t];
                                                            if (work = [t])
                                                                work = [t] + [t];
                                                                if (work = [t])
                                                                    work = [t] + [t];
                                                                    if (work = [t])
                                                                        work = [t] + [t];
                                                                        if (工作 = [t])
                                                                            work = [t] + [t];
                                                                            if (工作 = [t])
                                                                                work = [t] + [t];
                                                                                if (工作 = [t])
                                                                                    work = [t] + [t];
                                                                                    if (工作 = [t])
                                                                                        work = [t] + [t];
                                                                                        if (工作 = [t])
                                                                                            work = [t] + [t];
                                                                                            if (工作 = [t])
                                                                                                work = [t] + [t];
                                                                                                if (工作 = [t])
                                                                                                    work = [t] + [t];
                                                                                                    if (工作 = [t])
                                                                                                        work = [t] + [t];
                                                                                                        if (工作 = [t])
                                                                                                            work = [t] + [t];
                                                                                                            if (工作 = [t])
                                                                                                                work = [t] + [t];
                                                                                                                if (工作 = [t])
                                                                                                                    work = [t] + [t];
                                                                                                                    if (工作 = [t])
                                                                                                                        work = [t] + [t];
                                                                                                                        if (工作 = [t])
                                                                                                                            work = [t] + [t];
                                                                                                                            if (工作 = [t])
                                                                                                                                work = [t] + [t];
                                                                                                                                if (工作 = [t])
                                                                                                                                    work = [t] + [t];
                                                                                                                                    if (工作 = [t])
                                                                                                                                        work = [t] + [t];
                                                                                                                                        if (工作 = [t])
                                                                                                                                            work = [t] + [t];
                                                                                                                                            if (工作 = [t])
                                                                                                                                                work = [t] + [t];
                                                                                                                                                if (工作 = [t])
                                                                                                                                                    work = [t] + [t];
                                                                                                                                                    if (工作 = [t])
                                                                                                                                                        work = [t] + [t];
                                                                                                                                                        if (工作 = [t])
                                                                                                                                                            work = [t] + [t];
                                                                                                                                                            if (工作 = [t])
                                                                                                                                                                work = [t] + [t];
                                                                                                                                                                if (工作 = [t])
                                                                                                                                                                    work = [t] + [t];
                                                                                                                                                                    if (工作 = [t])
                                                                                                                                                                        work = [t] + [t];
                                                                                                                                                                        if (工作 = [t])
                                                                                                                                                                            work = [t] + [t];
                                                                                                                                                                            if (工作 = [t])
                                                                                                                                                                                work = [t] + [t];
                                                                                                                                                                                if (工作 = [t])
                                                                                                                                                                                    work = [t] + [t];
                                                                                                                                                    } ( )
```
What is a sequence of process execution that satisfies the safety constraint?

<table>
<thead>
<tr>
<th>Total</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>d^3</td>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>d^2</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>d^1</td>
<td>1</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>d^0</td>
<td>0</td>
<td>0</td>
<td>1</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 bankers

Example (contd)
and the sequence $P_0, P_2, P_3$ satisfies the safety constraint.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>I</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Allocation: 


Available: 

Max

3. The new system state after the allocation is: 

$P_0 \geq (1, 0) = (1, 0, 0)$, the maximum number $P_1$ can request. 

2. $(0, 2, 2) \geq (1, 2, 2)$, the available resources, and the maximum number $P_1$ can request. 

Yes, since the algorithm grants the request immediately. Show the system state, and other criteria.

If a request from process $P_1$ arrives for additional resources of $(0, 2)$, can the banker's algorithm grant the request immediately? Show the system state, and other criteria.

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.
  - Wake: Decrement a value. If thread continues if value $\geq 0$ (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: Controverts any one thread holds 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? The user?

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

Advantages

Disadvantages

Low-Level Synchronization Primitives: Hardware support

Synchronization Wrap up
Deadbolt avoidance
Deadbolt prevention
Deadbolt detection and recovery

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 - unblock and wake up any thread waiting in Acquire.
- Lock::Acquire - wait until lock is free, then grab 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 are execution states can a process be in? What do they mean? What causes a process to change execution states?
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
I. Things you should be able to do:

CPU Scheduling

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

3. What is preemptive scheduling? Why is non-preemptive scheduling? Which scheduling algorithms can be preemptive?

4. What is a time slice? What effect does a very small time slice have? What effect does a very large time slice have?

5. What is an I/O bound process? What is a CPU bound process? Is there any reason to prefer one to the other?

I. Topics you should understand:

CPU Scheduling

DEPARTMENT OF COMPUTER SCIENCE, UHARNESS
and does it avoid deadlock?

Guarantee mutual exclusion where appropriate, does it avoid starvation.

able to explain whether you believe it works. In particular, does it

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

Things you should be able to do:

**Synchronization**

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

9. What is the best? How can a test case instruction be used to support the

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

7. What is busy waiting?

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

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 prevent deadlock.

4. Given a state consisting of resources allocated to processes, maximum resource requirements of processes, and available resources, determine if the state could lead to deadlock.

3. Given a state consisting of resources allocated to processes, maximum resource requirements of processes, and available resources, determine if the processes are deadlocked.

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

Things you should be able to do:

- **Deadlocks**

Topics you should understand:

- **Deadlocks**
... system, such as Unix, Macintosh, Windows NT...

- You will not be asked detailed questions about any specific operating system, such as Unix, Macintosh, Windows NT.
- You will be asked to write pseudo code with synchronization.
- You should be able to read Java code.