CIS 415: Operating Systems

Prof. Allen D. Malony

Midterm – November 6, 2014

NAME: __________________________

<table>
<thead>
<tr>
<th>Section</th>
<th>Total Points</th>
<th>Points Scored</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Processes and threads</td>
<td>5 + 5 + 5 + 5</td>
<td></td>
</tr>
<tr>
<td>2. Synchronization</td>
<td>5 + 5 + 5 + 15</td>
<td></td>
</tr>
<tr>
<td>3. Scheduling</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4. Deadlock</td>
<td>5 + 5 + 15</td>
<td></td>
</tr>
<tr>
<td>5. Main memory</td>
<td>5 + 5 + 5</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20 + 30 + 15 + 25 + 15 = 105</strong></td>
<td></td>
</tr>
</tbody>
</table>

Comments:

1. Take a deep breathe.
2. Write your name on the front page now and initial all other pages.
3. Do all problems. Concept questions are each worth 5 points. In total, they are worth 57% of the grade!
4. Concept questions are intended to be short answer. If you are spending more than 5 minutes on any concept questions, you are writing too much! Move on!
5. If you have a question, raise your hand and we will come to you, if we can. Otherwise, we will acknowledge you and you can come to us.

Exams should be challenging learning experiences. Enjoy it!!!
1 Processes (20)

There are only concept questions in this section.

1.1 Concepts (20)

What is a process? What is thread? How do they differ?
A process is a program in execution. A thread is “thread of control” in the execution of a program. A thread
is part of a process and shares the process address space, files, and other resources, but each thread has its own
program counter, CPU registers, and stack.

What happens during a context switch?
Three things: 1) the state of the running process (program counter, CPU registers, memory management infor-
mation, ...) is saved into the process control block; 2) the state of the next process to be executed is loaded; and
3) the next process begins execution.

What is a process control block and what is its purpose?
The PCB holds all information necessary to save the state of the process (the “process context”) when it is context
switched off the processor and then context switched back on the processor and restarted.

What do the fork(), exec(), and clone() system calls do?
fork() creates a new process by duplicating the calling process. The new (child) process is an exact duplicate
of the calling (parent) process and has its own unique process ID. exec() replaces the current process image
with a new program executable. The idea is to have the child process run another program. clone() creates a
new process (like fork()), but the child process share parts of its execution context with the calling process, in
particular, the memory space.
2 Synchronization (30)

2.1 Concepts (15)

What does mutual exclusion mean?
Mutual exclusion is a condition on access to a resource that could be used by several processes. The condition states that only one process can access (hold) the resource.

What is a critical section? List the requirements for a solution to the critical section problem?
A critical section is a piece of code that should only be execute by 1 process (thread) at a time, usually because it accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution.

The requirements for a solution to the critical section problem are:
- mutual exclusion – only one process can be executing in its critical section
- progress – the decision as to which process is allowed to enter its critical section next is made in a finite time
- bounded waiting – a process will be allowed (guaranteed) access to the critical section after a finite waiting time

What is the difference between busy waiting and blocking?
With busy waiting, a process continuously loops waiting for access to a critical section.
With blocking, a process is blocked (context switched) when it must wait upon access to a critical section and is placed on a queue of waiting processes.
### 2.2 One for All and All for One (15)

A *barrier* is a synchronization mechanism used to guarantee that all threads reach a common synchronization point in their execution before any thread is allow to proceed further. Barriers are often used to satisfy a number of data dependencies simultaneously. Informally, each thread participating in the barrier performs the following general operations:

```plaintext
barrier:
    if (THIS thread is the LAST thread to arrive)
        then
            notify the other threads somehow that they can proceed executing
        else {
            indicate somehow that THIS thread has arrived at the barrier
            wait until notified somehow by the LAST thread to continue executing
        }
```

A “one-shot” barrier is a barrier that is intended to be used only once. The following pseudo-code (shown for one thread) outlines a partial solution to a “one-shot” barrier assuming that there are exactly \( N \) threads participating in the barrier. A non-blocking (“busy waiting”) binary semaphore is being used. Complete the solution (i.e., replace “???????” with the appropriate code).

The solution is to use the semaphore properly as below:

```plaintext
int count = 0; /* shared count variable */
barrier_sem = new Semaphore(1); /* shared barrier semaphore set initially to 1 */

void barrier() {
    wait(barrier_sem); /* need to lock around updates to count */
    count = count + 1;
    if (count == N)
        then {
            count = 0;
            signal(barrier_sem); /* unlock after count updates */
        }
    else {
        signal(barrier_sem); /* unlock after count updates */
        while (count > 0) ;
    }
}
```

Often several barrier synchronizations are performed between cooperating threads during a program’s execution. In this case, it is preferred to reuse, rather than replicate, the data structures and methods used to implement a barrier. Is the barrier implementation above reusable? That is, can the `barrier()` function be called again with the `count` and `barrier_sem` variables as set by the last barrier synchronization. Why or why not? Explain.

No, the barrier implementation above is not reusable. Consider the last process arriving at the barrier. If it sets `count` to 0 and then re-enters the barrier before the other processes test `count`, all processes will be busy waiting forever.
3 Scheduling (15)

There are no concept questions in this section, only a problem.

3.1 Who is the best? (15)

Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead. For Round Robin, assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For the others, assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound. Provide your answer in minutes.

Round Robin

0 ≤ t ≤ 10: each job gets 1/5 of CPU, at t = 10 C finishes
10 < t ≤ 18: each job gets 1/4 of CPU, at t = 18 D finishes
18 < t ≤ 24: each job gets 1/3 of CPU, at t = 24 B finishes
24 < t ≤ 28: each job gets 1/2 of CPU, at t = 28 E finishes
28 < t ≤ 30: each job gets 1/1 of CPU, at t = 30 A finishes

mean process turnaround time: 22 minutes

Priority Scheduling
finishing times: 6 (B), 14 (E), 24 (A), 26 (C), 30 (D)
mean process turnaround time: 18.8 minutes

FCFS (assumed arrival order A, B, C, D, E)
finishing times: 10 (A), 16 (B), 18 (C), 22 (D), 30 (E)
mean process turnaround time: 19.2 minutes

SJF
finishing times: 2 (C), 6 (C), 12 (B), 20 (E), 30 (A)
mean process turnaround time: 14 minutes
4 Deadlocks (25)

4.1 Concepts (10)

What are the necessary conditions for a deadlock to occur?
1. mutual exclusion
2. hold and wait
3. no preemption
4. circular wait

In deadlock avoidance, what is meant for the system to be in a safe state?
A system is in a safe state if there exists a sequence of resource assignments to processes such that all processes will terminate.
4.2 Robin Hood Resource Allocation (15)

One day, while reading about the infamous Robin Hood and his Merry Humans, you suddenly got an idea for a new resource allocation policy. After changing into your green tights, you define the rules for the policy. You assume that requests and releases for resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then any processes that are blocked, waiting for resources, are checked. If they have the desired resources, then these resources are taken away from this “rich” process and are given to the “poor” requesting process. The vector of resources for which the waiting process is waiting is increased to include the resources that were taken away.

Ok, good enough. Now you make up an example. Consider a system with three resource types and the vector $\text{Available}$ initialized to $(4,2,2)$. If process $P_0$ asks for $(2,2,1)$, it gets them. If $P_1$ asks for $(1,0,1)$, it gets them. Then, if $P_0$ asks for $(0,0,1)$, it is blocked (resource not available). If $P_2$ now asks for $(2,0,0)$, it gets the available one $(1,0,0)$ and one that was allocated to $P_0$ (since $P_0$ is blocked). $P_0$’s $\text{Allocation}$ vector goes down to $(1,2,1)$, and its $\text{Need}$ vector goes up to $(1,0,1)$.

Can deadlock occur? If so, give an example. If not, which necessary condition cannot occur?

Here is the series of states of the system starting from the beginning:

<table>
<thead>
<tr>
<th>Available</th>
<th>Process</th>
<th>Allocation</th>
<th>Need</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(4,2,2)$</td>
<td>P0</td>
<td>$(0,0,0)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>$(0,0,0)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>$(0,0,0)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
</tbody>
</table>

$P_0$ requests $(2,2,1)$

<table>
<thead>
<tr>
<th>Available</th>
<th>Process</th>
<th>Allocation</th>
<th>Need</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(2,0,1)$</td>
<td>P0</td>
<td>$(2,2,1)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>$(0,0,0)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>$(0,0,0)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
</tbody>
</table>

$P_1$ requests $(1,0,1)$

<table>
<thead>
<tr>
<th>Available</th>
<th>Process</th>
<th>Allocation</th>
<th>Need</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1,0,0)$</td>
<td>P0</td>
<td>$(2,2,1)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>$(1,0,1)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>$(0,0,0)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
</tbody>
</table>

$P_0$ requests $(0,0,1)$

<table>
<thead>
<tr>
<th>Available</th>
<th>Process</th>
<th>Allocation</th>
<th>Need</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1,0,0)$</td>
<td>P0</td>
<td>$(2,2,1)$</td>
<td>$(0,0,1)$</td>
<td>blocked</td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>$(1,0,1)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>$(0,0,0)$</td>
<td>$(-,-,-)$</td>
<td>not blocked</td>
</tr>
</tbody>
</table>
$P_2$ requests (2,0,0)

<table>
<thead>
<tr>
<th>Process</th>
<th>Allocation</th>
<th>Need</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>(1,2,1)</td>
<td>(1,0,1)</td>
<td>blocked</td>
</tr>
<tr>
<td>P1</td>
<td>(1,0,1)</td>
<td>(-,-,-)</td>
<td>not blocked</td>
</tr>
<tr>
<td>P2</td>
<td>(2,0,0)</td>
<td>(-,-,-)</td>
<td>not blocked</td>
</tr>
</tbody>
</table>

Deadlock will not happen because the *circular wait* condition cannot occur. Consider the scenario when where 2 of the processes are blocked. Because a process can satisfy a request either from the available resources or from blocked processes, the last unblocked process will always be able to meet its needs.

**Can indefinite blocking occur?**

Yes, it is possible that indefinite blocking can occur because there is nothing that guarantees that a process will not always encounter the unfortunate continuing circumstance where none of the other processes are blocked, but there are not enough available resources to satisfy its request.
5 Memory Management (15)

There are only concept questions in this section.

5.1 Concepts (15)

What is the difference between logical and physical memory?
Logical memory is the memory space “logically” used by a process during its execution.
Physical memory is the actual memory “physically” available in a computer system.

Why is address translation necessary? Define the purpose of the TLB hardware.
Address translation is necessary to convert logical addresses used by a process into physical addresses used to access physical memory.
The TLB is used in the memory management hardware to provide fast lookup of pages in a cache of the process page table.

What is paging and why is it useful?
Paging is a type of memory management whereby virtual memory is partitioned into fixed size pages and physical memory is partitioned into fixed size frames. Pages and frames are of equal sizes. Pages are assigned to frames as a process addresses virtual memory. Paging is useful because the fixed size partitioning of virtual and physical memory allows for efficient memory management with no external fragmentation and for development of hardware to support fast page address translation.