CS 345: Operating Systems Test 2

Professor Ellen Walker
Fall 2000 12 Week

NAME: ____________________________________________________________ KEY

Answer all questions in the space provided. Use the backs of the pages if you need extra space. You may have a single-page one-side crib sheet for this exam.

Please make sure that you have all 5 pages before beginning the test.

If you are stuck on a question, move on and come back to it later.

<table>
<thead>
<tr>
<th>No.</th>
<th>Pts.</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

1
1. (12 points) One way that semaphores can be implemented is by enabling and disabling interrupts (as is done in Nachos).

a. This technique can affect the system’s I/O in an adverse way. Explain.

While interrupts are disabled, I/O devices cannot inform the CPU that information is ready. If their buffers overflow, information is lost.

b. This technique does not work on multiprocessor systems. Explain.

Each processor has its own interrupts. Therefore, any processor whose interrupts are not disabled can execute the critical path. Simultaneously disabling interrupts on all processors would not be possible.

2. (12 points) Another way of implementing semaphores is "Busy Waiting".

a. Write a brief code fragment that exemplifies busy waiting. Explain how the busy waiting works.

```c
while(p<1) ;   //loop does nothing until p gets to 1
```

b. Why is busy waiting not the recommended solution for user-level process synchronization?

Busy waiting wastes the time of the CPU which could be better used executing non-waiting processes. User-level processes are likely to be waiting for a long time.
3. (15 points) For each of the following code segments, where the variables are initialized as shown, and threads A and B can run concurrently, what are the possible ending values of x?

a. int x = 5;
   int y = 17;
   semaphore m = 1;
   semaphore s = 0;
   thread A {
     x = x + 1;
     P(m);
     V(m);
   }
   thread B {
     x = y + 1;
     P(m);
     V(m);
   }

Ending values are: 6, 18, 19

If A and B are running in parallel and A loads, then B adds and stores, then
A adds and stores, the result is 6. Case b avoids this by adding mutual
exclusion, so the answer is 18 if A goes first and 19 if B goes first. Case
c adds synchronization, forcing B to go first (since m = 1).

b. (same variables)

(15 points) Fill in "Yes" or "No" in each empty column of the following table describing scheduling algorithms

<table>
<thead>
<tr>
<th>Algorithm name</th>
<th>Pre-emptive?</th>
<th>Starvation possible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Come First Serve</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Shortest Job First</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Round Robin</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Shortest Remaining Time</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
5. (20 points) The following questions deal with locks and condition variables.

a. Can you have a lock without a condition variable, a condition variable without a lock, both or neither? Explain.

You can have a lock without a condition variable (it is just like a semaphore), but you cannot have a condition variable without a lock. A condition variable is a structure that allows a process to release a lock while it sleeps (waits for a condition) and acquires the lock again when it wakes.

b. Describe what the condition::signal() function does.

The condition::signal function awakens the first waiting thread on the condition variable's queue and moves it to the ready queue so that it will execute when its turn comes. (At that time, it will re-acquire its lock).

c. In Nachos, waiting for a condition is done in a loop. What must the loop test for each time through?

The loop tests to be sure that the lock was successfully acquired, i.e. that the current thread is holding the lock that is associated with the condition variable.

d. Describe in detail a situation in which the wait in the loop of part c would be executed more than once.

If the condition::signal() function is sent to a thread, but while the thread is in the ready queue, another thread acquires the associated lock, then the thread must wait again.
6. (20 points) A sequence of processes' arrival times and burst times is given below. Compute
the average waiting time for each of the following algorithms. Suggestion: draw a Gantt chart
or otherwise show your work for partial credit.

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

a. First Come First Serve

0-3, P1; 3-13, P2; 13-19, P3; 19-21 P4
wait time = (0-0 + 3-1 + 13-2 + 19-4) / 4 = (2+11+15)/4 = 28/4 = 7

b. Shortest Job First

0-3, P1; 3-9, P3, 9-11, P4, 11-21 P2
wait time = (0-0 + 3-2 + 9-4 + 11-1) / 4 = (1+5+10)/4 = 16/4 = 4

c. Round Robin (time slice = 4)

0-3, P1[0]; 3-7, P2[6]; 7-11, P3[2],11-13; P4[0], 13-17, P2[2], 17-19,P3[0], 19-21 P2[0]
wait time = (0-0 + 3-1 + 7-2 + 11-4 + 13-7 + 17-11 + 19-17)/4 =
(2+5+7+6+6+2)/4 =28/4 = 7

d. Shortest Remaining Time

0-3, P1[0], 3-4 P3[5], 4-6 P4[0], 6-11 P3[0], 11-21 P2[0]
wait time = (0-0 + 3-2 + 4-4 + 6-4 + 11-1)/4 = (1+2+10)/4 = 3.25

(5 points) Two additional scheduling algorithms are Multilevel Queue Scheduling and
Multilevel Feedback Queue Scheduling. Explain the difference between them. Do not use the
word "feedback".

Both methods have several queues, each running at a different priority, so that
jobs from the lower-priority queues won't run until the higher priority queues
are empty. In Multilevel Queue scheduling, once a job enters a queue it
remains in that queue until it is finished. In Multilevel Feedback Queue
scheduling, processes can be upgraded to higher queues as they age.