CS 374 – Operating Systems NAME\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exam I

**This exam is worth 104 points. Please write your answers in the space provided—you can continue on the back of a page if necessary, and you may cross-reference to your prewritten answers rather than recopying the answer to a question on the first two problems. The Multiple Choice section is worth 32 points, so some of you may want to start there.**

1. Consider the following solution for the Dining Philosophers Problem, where philosophers **p** are processes and forks **f** are resources. Function rand() returns a value between 0 and 1, and in effect is a fair coin flip. (10 points)

p(i) {

while (1) {

think(i); **// if hungry, philosopher exits think**

if ( rand( ) < .5 ) { **// flip fair coin**

P(f[i]);

if ( P(f[i+1%N]) {

eat;

V(f[i]);

V(f[i+1%N]); }

else

V(f(i);

}

else {

P(f[i+1%N]);

if ( P(f[i]) {

eat;

V(f[i+1%N]);

V(f[i]); }

else

V(f(i+1%N);

}

}

}

* 1. Assume that the if-test works properly and allows one to test for having grabbed a fork, explain clearly how this solution avoids deadlock.

* 1. Give a repeating sequence of philosophers eating which results in starvation of a philosopher.

1. Consider the analogy of a tunnel with only a single lane. To avoid a deadlock, cars must be prevented from entering the tunnel at both ends simultaneously. Once a car enters, other cars from the same direction may follow immediately. Ignoring the problem of starvation, write the code using semaphores to solve this problem. (8 points)
2. The following is a process ordering specified in S & P notation:

S( S( P( p1, S(p2, p3)), p4), P(p5, p6)) (10 points)

* 1. Draw the equivalent process flow graph.
  2. Either express the graph using nested cobegin/coend blocks (no semaphores), or explain clearly why this is not possible.

1. Convert the following process flow graph into parallel code using a single cobegin/coend block with processes coordinated via semaphores (8 points):

p3

p6

p2

p4

p1

p5

p7

p8

1. Below is a program expressed using *fork*, *join*, and *quit*.
   1. Draw the process flow graph for the program.
   2. Either express the program using S/P notation, or explain clearly why this is not possible.

(10 points)

T1 = 3; T2 = 3;

fork L1; fork L2; quit;

L1: fork L4; p1; join T1, L6; quit;

L2: p2; fork L3; fork L5; quit;

L3: p3; join T2, FIN; quit;

L4: p4; join T1, L6; quit;

L5: p5; join T1, L6; quit;

L6: fork L7; p6; join T2, FIN; quit;

L7: p7; join T2, FIN; quit;

FIN: …

1. Let q be the millisecond size of a quantum, s be the size of a context switch, and c be the average wall clock time between interrupts for a process in the system. (10 points)
   1. If the useful CPU time for applications is 75% of the total CPU time, write an equation for s in terms of q.
   2. If the average process uses 3.5 quanta before a trap or interrupt occurs, write an equation for c in terms of q and s.
   3. Combine part a. and b. to write an equation for c in terms of s.
2. Below is a program expressed with S/P notation. Express this equivalently using cobegin/coend notation (*no semaphores*). (6 points)

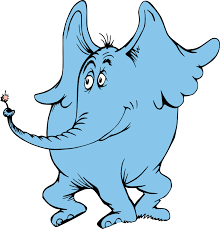
P( S( p5, P( p3, S( p2, p4 )), P( p1, S( p6, p8 ))), p7 )

1. Below is code for a Hoare monitor. Fill in the values of the variables m and n and the line numbers executed in order assuming that the functions are called in the following order: hhw.B( ), hhw.A( ), hhw.A( ), hhw.C( ), hhw.B( ). Take care to show all numbered lines executed in the proper order; the first column is filled in for you. (10 points)

**monitor** hhw { // “The Hoare-ton Hears a Who Monitor”

int m=5, n=5;

condition x, y, z;

 A( ) {

(1) m = m \* 2;

(2) if (m < n) x.wait;

(3) y.signal;

}

B( ) {

(4) if (n == m) y.wait;

(5) else { z.signal; }

(6) m = m - 3; // not part of else

}

C( ) {

(7) x.signal;

(8) n = n + 2;

(9) y.signal; z.wait;

(10) m = m / 2;

}

} // end monitor

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Line # =** | 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **m =** | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **n =** | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

I believe the number of boxes above is correct, but please use more or fewer if I am wrong.

**Multiple choice.** (2 points each.) Circle the letter of the correct response. *Be clear which one you are circling; if I can’t tell, your answer is wrong!*

1. On the previous problem, which of the following is an example of an execution difference that would happen using notify instead of signal with Java/Mesa semantics?
2. line 6 would never be executed
3. lines 10 and 6 would reverse order for hhw.C() followed by hhw.B()
4. the first hhw.A() will wake the first hhw.B() on line 3
5. the second hhw.A() will wake the first hhw.B() on line 3
6. for the sequence of calls given, there is no difference

**Problems 10-12 use the following information: Assume a processor has five jobs that require the CPU cycles listed in the table below. For priority, assume that the higher the number, the higher the priority that job has. Assume the RR time quantum is 4.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Arrival Time:** | **0** | **3** | **4** | **6** | **11** |
| **Job:** | **A** | **B** | **C** | **D** | **E** |
| **CPU Cycles:** | **10** | **3** | **1** | **8** | **4** |
| **Priority:** | **1** | **5** | **4** | **2** | **3** |

1. The time line shown below belongs to which process-scheduling algorithm?

|  |  |
| --- | --- |
| Job | 0 10 11 14 18 26 |
| A | **--------------------------** |
| B | **----------------** |
| C | **---------** |
| D | **------------------------------** |
| E | **------------------** |

* 1. First Come, First Served
  2. Shortest Remaining Time
  3. Round Robin
  4. Shortest Job Next
  5. Priority Scheduling

1. The time line shown below belongs to which process-scheduling algorithm?

|  |  |
| --- | --- |
| Job | 0 3 4 5 7 14 18 26 |
| A | **-------- -------------------------** |
| B | **----- --------** |
| C | **-----** |
| D | **------------------------------** |
| E | **------------------** |

* 1. Multiple Level Queues (ML)
  2. Shortest Remaining Time
  3. Round Robin
  4. Shortest Job Next
  5. Priority Scheduling

1. The time line shown below belongs to which process-scheduling algorithm?

|  |  |
| --- | --- |
| Job | 0 4 7 8 12 16 20 24 26 |
| A | **----------- --------------- -------** |
| B | **-----------** |
| C | **------** |
| D | **----------------- -----------------** |
| E | **-----------------** |

1. Biggus Thingus
2. Shortest Remaining Time
3. Round Robin
4. Shortest Job Next
5. Priority Scheduling
6. Spooling allows \_\_\_
7. multiple devices to synchronize with a master process controlling them.
8. multiple processes to print concurrently.
9. multiple sections of computer tape to be wound on the same spindle.
10. single sections of computer tape to be affixed to one another, end to end.
11. threads to be saved for later sewing.
12. Mechanisms other than semaphores and fork/join operations are considered necessary for synchronization because \_\_\_
13. not all processes we want to synchronize can share memory.
14. semaphores and fork/join are low level mechanisms; sometimes we want mechanisms that are easier to use and maintain.
15. processes on multiple machines may share a resource, like a printer, but only communicate using messages.
16. All of the above.
17. None of the above.
18. Two children are fighting while brushing their teeth. One holds the toothpaste, but refuses to give this up until receiving the toothbrush. The other holds the toothbrush, but refuses to give that up until getting the toothpaste. This is an example of \_\_\_\_

a. deadlock.

b. starvation.

c. a race condition.

d. a livelock.

e. a process control blocking.

1. *Abstraction* is a concept that we chiefly employ to \_\_\_\_\_\_\_\_\_\_\_\_\_
2. share resources.
3. manage resources efficiently.
4. do yard work, like mowing the lawn.
5. control system complexity.
6. send items to a printer.
7. The biggest difference between threads and processes is that threads ***do not*** have:

a. State: that is, their own program counters and copies of variables

b. Their own resources; they share resources with the process’s other threads

c. CPU time; they are never executed

d. Integer variables

e. Deadlock

1. The operating system kernel is

a. the part of the operating system which handles core functionality.

b. the secure part of the operating system; the rest is unsafe.

c. the hard nut at the center—the rest is chewy.

d. the other systems that this operating system communicates with.

e. what remains when you wipe a hard drive.

1. The role of an operating system is to
2. management the computer’s devices.
3. providing the user with a simpler view of the system hardware.
4. using the computer most efficiently.
5. coordinating resource usage by processes.
6. all of the above.
7. Atomicity is crucial to the join operation because without it we could have \_\_\_\_ on the join variable.
8. starvation
9. a deadlock
10. a race condition
11. an infinite value
12. a livelock
13. A Process Control Block (PCB) \_\_\_

a. typically has pointers to the code segments a process is running.

b. typically has pointers to the data used by a process.

c. contains information about a process’s status, resources, and priority.

d. has a memory map for its process.

e. all of the above.

1. When total CPU utilization U <= 1, what is true for EDF?
2. Processes will never wait for resources.
3. Processes that use less than U/10 can be scheduled.
4. The CPU quantum will be variable.
5. Any schedule meeting this is feasible.
6. CPU time can be balanced with memory utilization.
7. The foreach construct in C shell scripting languages executes its body of instructions over a list of files, directories, or accounts. This an example of
8. data parallelism
9. control parallelism
10. semaphore-controlled parallelism
11. a control alternative to loops that lacks parallelism
12. a resource management construct used with a when clause
13. What is one limitation to using the cobegin/coend construct?
14. It won’t work for shared memory parallelism.
15. It won’t translate into an equivalent fork/join construct.
16. It won’t represent all precedence graphs.
17. It won’t translate into an equivalent semaphore representation.
18. It won’t synchronize processes.

**PLEASE MAKE SURE YOUR NAME IS ON THE FRONT OF THE EXAM.**

**A SMALL AMOUNT OF EXTRA CREDIT IS BUILT IN TO THE TEST!**

#### HAVE A GREAT DAY!