FINAL EXAMINATION – SPRING 2003
CSC 262 – OPERATING SYSTEMS
NICHOLAS R. HOWE

This is a closed-book exam. You may use two double-sided 8.5x11 sheets of notes.

All answers to this exam should be written in your exam booklet(s). Start with the questions that you know how to do, and try not to spend too long on any one question. Partial credit will be granted where appropriate. You will have two hours and twenty minutes. Good luck!

1. **Vocabulary**. (16 points) Identify the term defined in class that matches the following definitions. If the answer is an acronym, specify what it stands for.

a. A cache that stores the page table entry for recently accessed pages.

TLB (Translation Lookaside Buffer)

b. A page replacement algorithm that measures how much pages are used and replaces ones that aren't used much.

LFU (Least Frequently Used)

c. A rule of thumb suggesting that for a memory system of variable-sized partitions in equilibrium, there should be twice as many partitions as holes.

Knuth's 50% Rule

d. This piece of hardware attached to the CPU translates virtual memory names into physical addresses.

MMU (Memory Management Unit)

e. This is the name for the process whereby an I/O device transfers data to or from the main memory, bypassing the CPU.

DMA (Direct Memory Access)

f. This principle or property of an executing program suggests that looking at the behavior in the recent past will indicate the likely behavior in the near future.

Locality

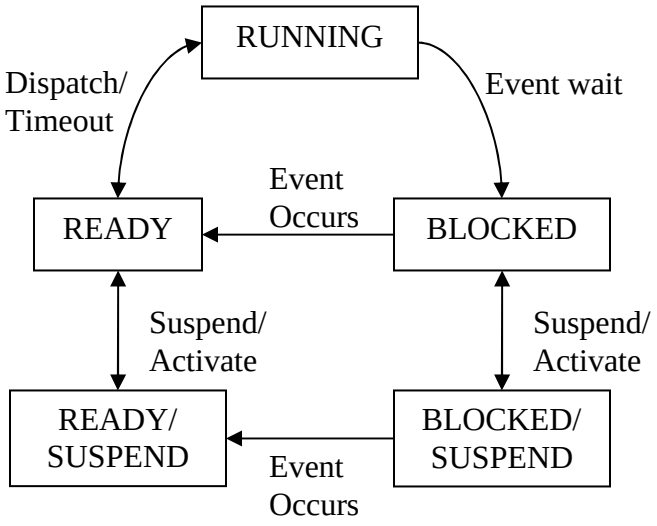
g. This term refers to a set of disk sectors that are simultaneously accessible without moving the read/write head, because they are located on corresponding tracks on each platter.

Cylinder

h. A software construct, defined to have nonnegative integral values, with two possible operations, sometimes called *wait* and *signal*.

Semaphore

2. **Process State**. (8 points) A particular operating system will have five process states: RUNNING, READY, BLOCKED, BLOCKED/SUSPEND, and READY/SUSPEND. (The latter two states apply to processes whose working set is not currently in memory.) Draw a transition diagram showing the allowable transitions between states. Label each transition with the mechanism that would cause that transition to take place.



3. **Race Conditions**. (12 points) For each of the following programs, state the possible values for *x* and *y* if the program terminates. Express your answer as a set of *(x,y)* pairs. For example, the starting configuration should be written as {(8,3)}.

In addition, state whether the program will always terminate, sometimes terminate, or never terminate.

You may assume that the variables are initialized as follows:

x : integer **init** 8
y : integer **init** 3
m0 : general semaphore **init** 0
m1 : general semaphore **init** 1

a). **cobegin** x := x-y; // y := y-x; **coend**

{(5,-5),(5,-2),(13,-5)}; always terminates

b).
```
cobegin
   DOWN(m1); x := x-y; UP(m1);
      //
   DOWN(m1); y := y-x; UP(m1);
coend
```

{(5,-2),(13,-5)}; always terminates

c).
```
cobegin
   DOWN(m0); x := x-y; UP(m0);
      //
   DOWN(m0); y := y-x; UP(m0);
coend
```

never terminates

d).
```
cobegin
   DOWN(m1); x := x-y; UP(m1); UP(m0);
      //
   DOWN(m0); y := y-x; UP(m0); UP(m1);
coend
```

{(5,-2)}; always terminates

4. **Paging and Latency**. (8 points) A large employee database is implemented on a system with 64-bit virtual memory. As a result, the employee records can be stored within the virtual address space as an array of records, in order according to the employee ID number. A record may be accessed by its ID number, or by social security number (SSN). Both access methods use hash tables and are similar in speed. Your boss wants you to write a function that will compute the total salary for all workers. Does it matter which method you use for access? Why or why not? (Assume that employee ID numbers are based upon the hire date and are thus unrelated to SSN, and that several records fit onto a page in memory.)

Working in order of employee ID number will be faster because it will provoke fewer page faults.

5. **Linux Processes** (8 points). Explain how to set and/or modify the priority of a process in Linux, assuming the kernel is using the `SCHED_OTHER` protocol. What limitations are placed on ordinary users (as opposed to root) as far as the ability to set process priorities?

Priorities are represented by a number from -19 to +20, and default to 0. The **`nice`** *command may be used to run a process at a different priority level. Only root may run a process at priority less than 0. The* **`renice`** *command adjusts the priority of a running process. Only root may upgrade the priority (i.e., lower the number).*

6. **Virtual Memory**. (18 points) You're working on the memory manager for a new computer, code-named Elephant. Elephant will use a pure segmented memory system, with up to eight segments available. The segment to be used will be indicated by the first three bits of the virtual address. Segments are allocated in sizes that are multiples of 1K, and every segment must begin in memory at an address that is a multiple of 1K. The base and limit of the segment may then be given in the segment table with the last ten bits left off.

Below is a sample segmentation table for Elephant. For the memory actions given below, specify (i) which faults, if any, would occur as a result; (ii) specify the address in memory that would come out of the MMU, if any; (iii) specify any changes to the segment table that would occur, if any.

				Permissions			
Segment	*Resident*	*Dirty*	*Locked*	*Read*	*Write*	*Base*	*Limit*
000:	1	0	0	0	0	000000	1100
001:	1	0	0	1	1	100000	1111
010:	1	0	0	1	1	001100	1010
011:	1	0	0	1	0	010110	1001
100:	1	1	1	1	1	111100	0011
101:	1	0	0	1	1	110000	1000
110:	0	1	1	1	0	111100	0011
111:	0	0	0	1	1	110000	1000

Note: to compute a physical address from a virtual, read the segment number from the first three virtual bits. Take the base and limit from the appropriate segment descriptor and add ten zeros (1K) to each. The physical address will be the base plus the offset (the remaining 13 bits of the virtual address), unless the offset is greater than the limit.

a. Read operation to address `0011010101010101`.

(i) No faults
(ii) 1010101010101010
(iii) No changes

b. Read operation to address `1101000000001010`.

(i) Memory fault (segment not resident)
(ii) No address due to fault (would have been 10001000000001010)

(iii) No changes (segment cannot be brought into memory in current location because segment 100 is locked.)

c. Read operation to address `0000000000000010`.

(i) Permission fault (read not allowed)
(ii) No address due to fault (would have been 0000000000000010)
(iii) No changes

d. Write operation to address `0011010101010101`.

(i) No faults
(ii) 10 10101010101010
(iii) Dirty bit set

e. Write operation to address `1001110101010101`.

(i) Segmentation fault (limit exceeded)
(ii) 10010101010101010
(iii) No changes

f. Write operation to address `0110010101010101`.

(i) Permission fault
(ii) No address due to fault (would have been 0110001010101010)
(iii) No changes due to fault (would have been change to dirty bit.)

7. **Page Replacement**. (8 points) Slick Willy has proposed to install a new page replacement algorithm on your computer. He shows you the results of an experiment he did, where he counts the page faults generated by a particular process allocated a certain number of frames to run. The count shows the algorithm does nearly as well as OPT, and much better than LRU on this experiment. Assuming you believe the results of this test are true, what concerns might keep you from adopting the new policy despite such impressive results? (You should be able to give at least two separate considerations.)

First, the experiment was only carried out for a single process. It is easy to find an algorithm that does better than LRU for a given process. Conversely, one can find a process on which many replacement algorithms will do better than LRU. The important question is whether the performance will be better on a variety of typical tasks. Second, even if the policy is better than LRU with the number of frames in the experiment, there is no guarantee that its superiority will apply at different working set sizes.

8. **Bit Calculations**. (6 points) A newly designed operating system will use a segmented, paged virtual memory system (similar to the Multics virtual memory) with

32-bit virtual addresses. Each process will have access to up to 64 segments, and each segment will have its own page table.

a. How many bits are available for the page name and page offset combined? What is the maximum length of a segment, in words?

26 bits available $\Rightarrow$ maximum length 2^{26} words, or 64 MW.

b. Suppose that pages are $\boldsymbol{2^f}$ words in size. In terms of $\boldsymbol{f}$, what is the expected (average) internal fragmentation per segment?

$0.5*2^f = 2^{f-1}$

c. What is the expected table fragmentation per segment, again in terms of $\boldsymbol{f}$? Assume that each page takes one word to describe in the page table.

2^{26-f} *words per segment*

9. **Ext2 File System** (8 points). The diagram below shows the data structures maintained within a block group of the Ext2 file system. Identify all the data structures that would need to be used in order to complete each of the following tasks. Assume that nothing is cached in memory, so that all information for each part must be read in from the disk. (Note: each part will typically use several of the data structures below.)

SUPER BLOCK	GROUP DESCRIPTORS	BLOCK BITMAP	INODE BITMAP	INODE TABLE	DATA BLOCKS

a). Determine how much free space is available on the disk.
Super Block.

b). Locate a free inode and mark it used.
Super Block, Group Descriptors, Inode Bitmap

d). Check the modification date of the root directory.
Super Block, Group Descriptors, Inode Table

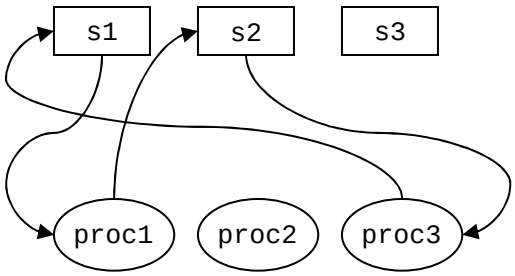
d). Read the contents of the root directory of the disk.
Super Block, Group Descriptors, Inode Table, Data Blocks

10. **Semaphores and Deadlock**. (8 points) Consider the following process definitions involving semaphores. Which set(s) of processes might become deadlocked? Draw resource allocation diagrams for any scenarios where deadlock is possible.

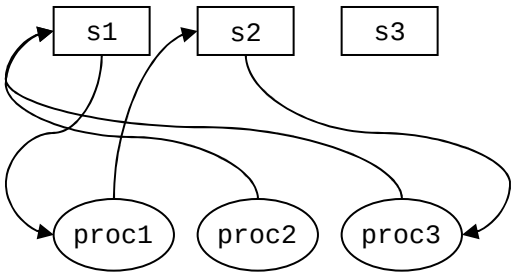
```
init:    semaphore s1(1), s2(1), s3(1);
```

```
proc1:  process
   DOWN(s1);  DOWN(s2); DOWN(s3); UP(s3); UP(s2); UP(s1);
endprocess;

proc2:  process
   DOWN(s1);  DOWN(s3); UP(s3); UP(s1);
endprocess;

proc3:  process
   DOWN(s2);  DOWN(s1); UP(s1); UP(s2);
endprocess;
```

Processes 1 and 3 may deadlock if `proc1` grabs `s1` and `proc3` grabs `s2`. All three processes may become deadlocked if the above scenario occurs and then `proc2` attempts to grab `s1`.



Scenario 1



Scenario 2