FINAL EXAMINATION – SPRING 2004
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. **Binary Numbers** (8 points). What is the minimum number of bits needed to represent a number of the given size? In other words, given **x**, what is $\lceil \mathbf{log_2(x)} \rceil$?

   a. 512 G
   b. 65536
   c. 32K*8K
   d. 106

2. **File Systems** (12 points). MiniFS-2 is a Unix-like file system built with extremely tiny disk blocks: only 8 bytes! Each MiniFS-2 disk has a maximum of 255 data blocks, so a pointer to a block takes exactly one byte. I-nodes in MiniFS have two direct pointers, one single indirect pointer, and one double indirect pointer, and no triple indirect pointers (in that order). The data blocks are numbered starting with 1, since a block number of 0 indicates a null pointer.

   a. How many block pointers will fit in a disk block? How many content blocks in total can be pointed to by the double indirect block alone?

   b. Given the i-node at right and data blocks shown below, what are the content blocks of the specified file? List all the content **data block numbers** in order. Do not include any pointer blocks or blocks belonging to other files.

i-node
34
254
3
6

Block 1	*Block 2*	*Block 3*	*Block 4*	*Block 5*	*Block 6*	*Block 7*	*Block 8*
56	168	230	116	203	4	165	159
164	150	247	226	209	0	15	123
248	122	14	0	30	0	105	112
32	105	166	0	115	0	72	60
3	67	50	0	170	0	123	62
162	208	110	0	48	0	4	242
193	29	145	0	27	0	211	19
252	209	251	0	135	0	200	241

3. **Virtual Memory** (15 points). Consider the (partial) page table below for a hypothetical operating system that uses 12-bit virtual addresses with pages of 256 B in size. Assume that 1K of physical memory is available on this system. For each action shown below, specify (i) any faults that would occur as a result (or "none" if applicable), (ii) the physical address that would be generated (or "none" if applicable), and (iii) the appearance of the corresponding line in the page table after the action is complete. You may make the following additional assumptions:

- Actions are not cumulative, e.g., each item refers to the table as shown below.
- Pages brought into memory will go into the highest-numbered free frame.
- Only the kernel may access pages with the system bit set.
- Any process may access pages with the system bit unset.
- No process may write to a read-only page.
- Pages are shown in order with page 0 at the top.

Resident	Dirty	Usage	System	Read-Only	Frame
0	1	0	0	0	11
0	0	1	1	0	01
1	0	0	0	1	11
1	0	1	1	0	00
1	1	1	0	0	01
0	1	0	1	1	00
…	…	…	…	…	…

a. User process reads address 001011001100.

b. User process writes address 001100000001.

c. Kernel process writes address 000100010001.

d. Kernel process reads address 001100000001.

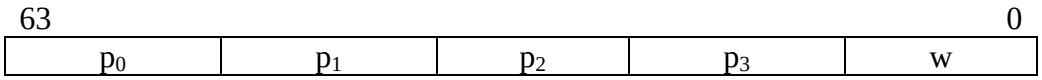
e. Kernel process writes address 010101010101.

4. **Memory Concepts**. In a few paragraphs, define **fragmentation** and explain how it affects the design of memory systems:

a. (12 points) Define three different types of fragmentation, and give an example of a memory design that might exhibit each one. List and explain measures that can minimize the amount of fragmentation in each of your example systems.

b. (9 points) Explain the similarities (if any) between memory fragmentation and fragmentation of the storage space on a hard disk, and examine the role played by each type of fragmentation in a filesystem such as **ext2**.

5. **Bit Computations** (12 points). You are helping to design an operating system that will use 64-bit addresses. The memory for this system will by byte-addressable, and the page size will be 16KB. The virtual memory will use four levels of page tables.



a. If a page descriptor requires 8B, how many such descriptors can fit onto one page?

b. If all page tables except the top level are designed to fit exactly onto a single page, how many bits are in $\mathbf{p_0}$, $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$, and **w**, respectively?

c. How much space will be required to store the main (top-level) page table?

6. **Cryptography and Operating Systems** (12 points). Capabilities can be implemented in an operating system using the technology of *digital signatures*: the kernel grants a capability to a process by providing a digitally signed message that the resource can verify when the process attempts to use it. Suppose that a particular operating system uses capabilities implemented via the Rabin signature scheme. Answer each of the following questions in a sentence or two.

a. What secret information (key) would the kernel need in order to grant a capability?

b. What information would the driver for a resource need in order to verify that a particular capability was indeed granted by the kernel and not forged?

c. How hard would it be for a rogue process to determine the kernel's secret key?

7. **Pentium** (8 points). The Intel Pentium line of chips is designed to operate using a variety of memory management schemes. In particular, it can support virtual memory using both segments and paging, just segments, just paging, or neither. Briefly outline how each of these conditions can be realized.

8. **Race Conditions and Deadlock** (12 points). For each of the following programs, state whether the program will always terminate, sometimes terminate, or never terminate.

Next, state all 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 {(3,5)}.

Finally, if the program will not terminate, draw a resource allocation diagram showing how the processes are deadlocked.

You may assume that the variables are initialized as follows:

x : integer **init** 3
y : integer **init** 5
m0 : general semaphore **init** 0
m1 : general semaphore **init** 1
m2 : general semaphore **init** 1

a).
```
cobegin
   DOWN(m1); x := y+1; UP(m2);
      //
   DOWN(m2); y := x-1; UP(m1);
coend
```

b).
```
cobegin
   DOWN(m1); DOWN(m2); x := y+1; UP(m2); UP(m1);
      //
   DOWN(m2); DOWN(m1); y := x-1; UP(m1); UP(m2);
coend
```

c).
```
cobegin
   DOWN(m1); DOWN(m2); x := y+1; UP(m1); UP(m0);
      //
   DOWN(m1); DOWN(m0); y := x-1; UP(m1); UP(m2);
coend
```