FINAL EXAMINATION KEY – SPRING 2004
CSC 262 – OPERATING SYSTEMS
NICHOLAS R. HOWE

1. **Binary Numbers** (6 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	*39*
b.	65536	*16*
c.	32K*8K	*28*
d.	106	*20*

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-2 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?

8B per block / 1B per pointer = 8 pointers per block.
Double indirect points to 8^2 = 256 blocks.

i-node
34
254
3
6

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.

34 and 254 are direct pointers to content blocks. 3 is a single indirect block, containing pointers to content blocks 230, 247, 14, 166, 50, 110, 145, 251. 6 is a double indirect block, containing just one pointer to a single indirect block at 4. 4 is a single indirect block containing pointers to just two content blocks, 116 and 226. The file content blocks are therefore 34, 254, 230, 247, 14, 166, 50, 110, 145, 251, 116 and 226.

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** (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 binary 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 one 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.

i. No faults.
ii. Address 1111001100.
iii. Usage bit set to 1.

1	0	1	0	1	11

b. User process writes address 001100000001.

i. Permission fault (page has system bit set).
ii. No address generated. (Would be 0000000001 in absence of fault.)
iii. No change to table.

1	0	1	1	0	00

c. Kernel process writes address 000100010001.

i. Page fault (page brought into frame 10).
ii. Address 1000010001.
iii. Resident bit set to 1, dirty bit set to 1, usage bit set to 0, frame set to 10.

1	1	0	1	0	10

d. Kernel process reads address 001100000001.

i. No faults.
ii. Address 0000000001.
iii. No change to table.

1	0	1	1	0	00

e. Kernel process writes address 010101010101.

i. Permission fault (page is read-only).
ii. No address generated.
iii. No change to table.

0	1	0	1	1	00

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**.

Fragmentation is defined as memory that is unusable to user processes due to the design constraints of the memory system. Three common types of fragmentation are internal, external, and table fragmentation. Typically, one or more of these can be minimized or eliminated at the expense of increasing the others.

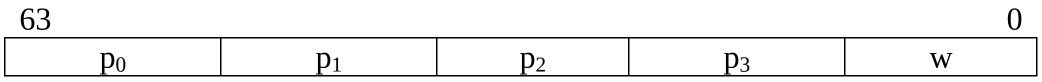
External fragmentation refers to the portion of memory that has been split into regions that are too small to be of any use. It typically occurs in segmented or partitioned memory systems where memory is repeatedly allocated and released in regions of arbitrary size. The policy used to choose where to allocate a new region of memory has some influence on the amount of external fragmentation. Memory compaction (moving allocated regions in memory to coalesce the unusable holes) can eliminate external fragmentation temporarily but is a time-consuming process.

Internal fragmentation refers to the portion of memory that has been allocated to a process in excess of that process's actual requirements. It is common in paged memory systems and other designs where memory cannot be allocated in arbitrary amounts, but must instead be quantized in multiples of some base amount. The average fragmentation per process is one half the page size, so decreasing the size of a page will reduce the amount of internal fragmentation.

Table fragmentation refers to memory that is unusable by user processes because it holds data required for the overhead and administration of the memory system. Simpler memory systems will tend to exhibit lower table fragmentation because they have lower overhead requirements, but the primary mechanism for reducing table fragmentation is to keep only the most necessary table data resident in memory.

All of the types of fragmentation we have examined in memory show up in disk storage systems as well. For example, ext2 maintains data structures such as the inode table, block bitmap, etc. that occupy space on the disk (table fragmentation). Space for files is allocated in sectors of fixed size, resulting in internal fragmentation. Finally, although the disk blocks for a particular file can be located in any order on the disk, and may be broken up into multiple separate pieces, this reduces the speed of operations on the file. Thus when external fragmentation of the free blocks on the disk becomes extreme, disk performance is impaired.

5. **Bit Computations** (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?

16KB / 8B = 2K

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?

p_0 = 17 bits, p_1 = 11 bits, p_2 = 11 bits, p_3 = 11 bits, w = 14 bits.

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

$(2^{17}\ \textit{entries})\cdot(8B / \textit{entry}) = 2^{20}\ B\ = 1MB$

6. **Cryptography and Operating Systems** (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?

The operating system would need two large distinct prime numbers of around 100 digits each.

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?

The driver would need to know the product of the two secret numbers.

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

Factoring the public key or guessing the secret numbers by chance would take longer than the lifetime of the universe using known algorithms if the prime numbers chosen are of sufficient size.

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.

Segments are specified by giving a base address and a limit, and may overlap. The current segments to use for code, data, and stack references are stored in registers. To effectively turn segmentation off, these registers can be loaded with a segment with base address 0 and limit equal to the size of the memory. Applying the base address of the segment generates a 32-bit linear address. The Pentium has a software switch (controlled by the operating system) that determines whether this is interpreted directly as a physical address (paging off) or as a virtual address in a paged memory system.

8. **Race Conditions and Deadlock** (12 points). For each of the following programs, 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)}.

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

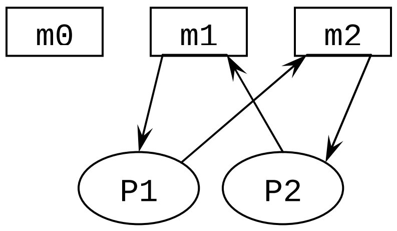
Finally, if the program will not terminate, draw a resource allocation diagram showing how the processes are deadlocked. In the diagram, refer to the first parallel thread as P1 and the second as P2.

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

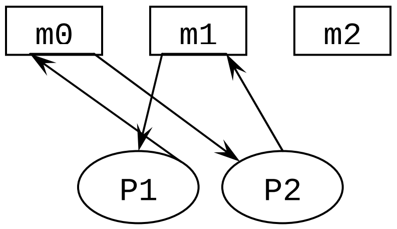
a).
```
cobegin
   DOWN(m1); x := y+1; UP(m2);
      //
   DOWN(m2); y := x-1; UP(m1);
coend
```

Always terminates. (x,y) ∈ {(4,4),(4,3),(5,4)}



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
```

Sometimes terminates. (x,y) ∈ {(4,3),(5,4)}



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
```

Sometimes terminates. (x,y) ∈ {(4,3)}