Name, SURNAME and ID ⇒

Middle East Technical University
Department of Computer Engineering

CENG 334

Section 1 and 2
Spring '2008-2009

FINAL

- **Duration: 120** minutes.
- **Exam:**
  - This is a **closed book**, **closed notes** exam. No attempts of cheating will be tolerated. In case such attempts are observed, the students who took part in the act will be prosecuted. The legal code states that students who are found guilty of cheating shall be expelled from the university for **a minimum of one semester**!
- **About the exam questions:**
  - The points assigned for each question are shown in parenthesis next to the question.
  - Whereever available, use the boxes to write down your answers.
- **This booklet consists of *8* pages including this page. Check that you have them all!**
- **GOOD LUCK !**

Question 1

Question 2

Question 3

Question 4

Question 5

Question 6

Total ⇒

1 (10 pts)

Consider the following solution to the **barrier** problem. In this solution `count` keeps track of how many threads have arrived. `mutex` provides exclusive access to `count` so that threads can increment it safely. `barrier` is locked (zero or negative) until all threads arrive; then it should be unlocked (1 or more).

In the barrier problem that we are trying to solve in this question, there are `n` student threads that run concurrently. We want to make sure that all `n` students have `arrived_at_the_door_of_BMB1` before allowing them to `get_in_BMB1`.

```
n = the_number_of_threads;                    // variables
count = 0;
mutex = Semaphore(1);
barrier = Semaphore(0);

arrive_at_the_door_of_BMB1();          // student thread code.

mutex.down();
    count = count + 1;
mutex.up();

if (count == n) barrier.up();

barrier.down();

get_in_BMB1();
```

Unfortunately, this solution is not quite right.

- What's the problem with this solution? Give an example on how it would occur.

- Fix the solution by changing (deleting/inserting/changing) the code provided above.

2 (20 pts)

For a computer with four page frames, the following reference string shows all the memory references for a certain process. This is a list of the virtual pages referenced (for a single process):

```
1 2 4 3 2 1 5 1 4 10 2 1 4 3 1 2 2 2 1 4
           ^^
```

When page 10 is to be accessed, the 4 page frames are full and one of the pages must be moved from memory to disk. You should assume there are 4 memory accesses per time interval and that all R bits get cleared at the end of each interval. For each algorithm listed below, indicate which virtual page will moved from memory to disk when page 10 is referenced. If it is possible that more than one page can be removed (a random choice is made), list all pages that could possibly be removed by the algorithm. Show your work in the free space but write down the result in the box.

- OPTIMAL

- FIFO

- LRU

- Clock (second chance). Assume that after the first time interval the clock hand is pointing to page frame 1.

3 (10 pts)

A virtual memory system that uses one-level pagins scheme has the following parameters: The pages are 2^P bytes long; virtual addresses are V bits long, organized as follows:

```
Virtual_Page_Number | Page_Offset
```

The page table starts at physical address $PTBL$; and each page table entry is a 4-byte word, so that, given a virtual address, the relevant page table entry can be found at $PTBL$ + (page number)*4. Answer the following in terms of the parameters P and V. Show your work in the free space but write down the result in the box.

- (1 pts) How many bits long is the *page offset* field?
- (1 pts) How many bits long is the *virtual page number* field?
- (2 pts) How many entries does the page table have, and what is the highest address occupied by a page-table entry?
- (3 pts) How many pages long is the page table?
- (3 pts) What is the smallest value of P such that the page table fits into one page?

4 (20 pts)

A virtual memory system uses a two-level paging scheme for address translation. Virtual addresses are 32 bits long, and the page size is 4096 (2^{12}) bytes. Each page table occupies a full frame, and each page table entry is 4 bytes long. The maximum physical memory size is 1 megabyte (2^{20} bytes).

- Draw an address translation diagram showing how addresses are translated in this system (you can neglect the TLB part). The diagram should show the sizes of the virtual and physical addresses, how they are broken down into fields, and how each field is used to derive the physical address. It should show the base register, any tables used for translation, and the maximum sizes of those tables.

- Suppose that a process' virtual address space is structured as in UNIX, with program code at the "bottom" of the address space, program data (the heap) immediately following the code, and the stack at the "top". Suppose that the program code occupies 2000 pages, the data occupies 1000 pages, and the stack occupies 100 pages in the address space - the remainder of the address space is unused. How many frames will be occupied by the page table(s) for this process?

5 (20 pts)

- Consider a UNIX-style i-node with 10 direct pointers, one single-indirect pointer, and one double-indirect pointer. Assume that the block size is 8 Kbytes, and that the size of a pointer is 4 bytes. How large a file can be indexed using such an i-node? Just write down the arithmetic expression, do not try to add them up.

- Consider a FAT-like file system that operated on a disk with 2^{10} sectors of 2^7 bytes each. The file system uses a block size of 4 sectors. A block pointer occupies two bytes. How many disk sectors are required to hold the FAT table?

- Why do file systems provide `open` and `close` operations? For UNIX file systems, describe the work that must be performed by the file system when an `open()` call is made. In particular, describe what changes are made to the file system's in-memory data structures.

- The free space on a disk can be stored using free-blocks or using a bitmap structure. Consider a disk that contains 2^{40} bytes with a 1KByte block size and 32-bit disk block numbers. Assuming the disk is completely empty, how many blocks would you need to store the free disk space using the free-block structure and using the bitmap structure? How would these values change when the the disk gets 80% full?

6 (20 pts)

Answer the following True/False questions. Mark your answers in the table provided below writing T(rue) or F(alse). Note that each wrong answer will be punished with -2 points.

a	b	c	d	e	f	g	h	i	j

(a) A virtual memory system that uses paging can suffer from internal fragmentation.

(b) In Unix-derived file systems, soft links are implemented using *link counts* in the file's inode.

(c) The logical dump of a filesystem on a disk always takes more time than the physical dump of the disk?

(d) The domain of a protection for a user is set at the time of his login and never changes.

(e) Assume that you are using the PGP-keys (Pretty Good Privacy) to secure the privacy of your communication with your friend. Your friend sends you an encrypted e-mail message. You can decrypt the message using his public PGP-key.

(f) The stack of a process can be paged-out to only swap area.

(g) The interrupt table of an OS is modified when a new device driver is loaded.

(h) A tape can only be introduced as a character I/O device to the OS.

(i) Buffering is used only when the application is receiving input data coming from the I.O device.

(j) Synchronization primitives that are implemented through enabling/disabling interrupts would also work in a multi-processor system.