

# Midterm #2

CMSC 412  
Operating Systems  
Fall 2005

November 22, 2004

## Guidelines

This exam has 7 pages (including this one); make sure you have them all. Put your name *on each page* before starting the exam. Write your answers directly on the exam sheets, using the back of the page as necessary. Bring your exam to the front when you are finished. Please be as quiet as possible.

If you have a question, raise your hand and I will come to you. Errors on the exam will be posted on the board if they are discovered. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn't need to do this at all, so be careful when making assumptions.

*Use good test-taking strategy:* read through the whole exam first, and answer the questions easiest for you first.

Question	Points	Score
1	15	
2	20	
3	25	
4	15	
5	25	
Total	100	

1. (Fragmentation, 15 points)

- (a) (6 points) Which of the following schemes for *memory allocation* exhibit internal fragmentation? Which exhibit external fragmentation? Check all the boxes that apply.

Allocation Scheme	External	Internal
Contiguous		
Segmented		
Paged		

- (b) (6 points) Which of the following schemes for *file allocation* exhibit internal fragmentation? Which exhibit external fragmentation? Check all the boxes that apply.

Allocation Scheme	External	Internal
Contiguous		
Linked		
Indexed		

- (c) (3 points) Of the schemes in part 1b that exhibit internal fragmentation, what is the maximum internal fragmentation possible per file if using a block size of  $B$  bytes?

2. (Page Replacement, 20 points)

- (a) (4 points) Suppose a program issues a request for a page not currently in main memory. Given that main memory is composed of 4 frames, use the following table to determine which frame's contents should be swapped out:

Frame #	Time When Loaded	Time Last Referenced	Referenced Bit	Modified Bit
0	126	279	0	0
1	230	280	1	0
2	120	282	1	1
3	160	290	1	1

- i. When using FIFO page replacement.
  - ii. When using LRU page replacement.
  - iii. When using Most-Recently Used (MRU) page replacement.
  - iv. When using Least-Frequently Used (LFU) page replacement.
- (b) For each of the next two questions, assuming no pages are in memory at the start, indicate which pages are in memory in the indicated frames *following* each reference in the sequence, using the given page replacement algorithm. If a fault occurs, put a check the “fault?” box. If an algorithm needs to evict a frame, but could choose equally from among multiple frames (i.e., there is a tie according to the algorithm), choose the frame for the page that was brought in to memory least recently. (Next page, please.)

i. (8 points) FIFO page replacement.

page reference	0	1	7	2	3	2	7	1	0	3
frame 1										
frame 2										
frame 3										
frame 4										
fault?										

ii. (8 points) LRU page replacement.

page reference	0	1	7	2	3	2	7	1	0	3
frame 1										
frame 2										
frame 3										
frame 4										
fault?										

3. (File System Implementation, 25 points)

**Background:** Say we have a file system that uses 1000 byte blocks. In the file system's metadata, block numbers are expressed as 2 byte integers.

Assume that all accesses to the disk are through a buffer cache that contains at most 4 blocks, using an LRU replacement strategy. Writes to disk should only occur when the user indicates this explicitly with a `sync()`, or when a dirty buffer needs to be evicted. Do not assume that you have any other caches in the system (e.g., a name cache).

The root directory is stored at block 1, with all directory entries fitting in this single block. This directory contains a file `file`, consisting of 10 data blocks. `file`'s FCB and data blocks are stored on disk at the following blocks:

File block #	FCB	0	1	2	3	4	5	6	7	8	9
Disk block #	10	4	12	15	16	17	2	3	20	21	22

**Problem:** Say a user program issues the following system calls for accessing the filesystem. Assuming we start with an empty buffer cache, fill out the table below indicating how many disk blocks are read or written on each system call. For the first column, assume you are using a file allocation table (FAT). Assume the FAT is stored at block 0 on the disk. For the second column, assume you are using a indexed allocation strategy, where the index block for `file` is stored at block 23.

If you wish, you may state assumptions about your answers, draw pictures, etc. It may be useful for you to keep a picture of the contents of the buffer cache next to each line in the table below. This may get you some partial credit. However, if these things are incorrect, you could lose points.

Operation	FAT		Indexed	
	# reads	# writes	# reads	# writes
open <code>/file</code>				
read 50 bytes at offset 100				
write 200 bytes at offset 400				
write 50 bytes at offset 1100				
write 50 bytes at offset 8100				
read 50 bytes at offset 250				
sync				
read 50 bytes at offset 3400				

4. (Disk Scheduling, 15 points)

(a) (5 points) In the last problem, the file system implementation only interacts with the buffer cache, and it is the buffer cache that actually reads from or writes to the disk. Using this scheme, when can the disk scheduling algorithm make a difference in system performance? Does it make any difference for this particular trace?

(b) (10 points) Assume you have a disk with 100 blocks (numbered 0–99), and the operating system's queue to read or write from the device contains the blocks 5, 50, 25, 95, 10. If the disk head is currently at block 45, what is the total *seek distance* when using

i. Shortest Seek Time First (SSTF)

ii. First-Come First-Served (FCFS)

5. (Short Answer, 25 points)

(a) (5 points) What is a virtual filesystem (VFS)? Why is it useful?

(b) (10 points) Say I have a paged memory system that uses a hash-table-based page table and a TLB, where the time  $\epsilon$  to access the TLB is .001 memory references, and page table lookups take, on average, 2 memory references (not including the actual read to the requested address). Say memory references hit the TLB 70% of the time, on average.

If as a hardware designer I had the choice to increase the size of the TLB to improve the hit rate to 80%, or to use a smarter hashing algorithm to reduce the average page table lookup time to 1.8 references, which choice should I make if I want to improve the effective access time? Justify your answer.

(c) (5 points) Can virtual memory be implemented using segmentation? Why or why not?

(d) (5 points) Would it make sense to implement a file system for a tape drive using standard techniques? Why or why not?