CS-537: Midterm Exam (Fall 2008)
Hard Questions, Simple Answers

Please Read All Questions Carefully!

There are seven (7) total numbered pages.

Please put your NAME and student ID on THIS page, and JUST YOUR student ID (but NOT YOUR NAME) on every other page. Why are you doing this? So I can grade the exam anonymously. So, particularly important if you think I have something against you! But of course, I don't. I really do want *all of you* to do well. Really! OK, most of you.

Name and Student ID: ____

Grading Page

	Points	Total Possible
Q1		25
Q2		25
Q3		25
Q4		25
Total		100

1. **Bit by Bit.** Assume you have a small virtual address space of size 64 KB. Further assume that this is a system that uses *paging* and that each page is of size 8 KB.

   (a) How many bits are in a virtual address in this system?

   (b) Recall that with paging, a virtual address is usually split into two components: a virtual page number (VPN) and an offset. How many bits are in the VPN?

   (c) How many bits are in the offset?

   (d) Now assume that the OS is using a *linear page table*, as discussed in class. How many entries does this linear page table contain?

   Now assume you again have a small virtual address space of size 64 KB, that the system again uses *paging*, but that each page is of size 4 bytes (**note: not KB!**).

   (a) How many bits are in a virtual address in this system?

   (b) How many bits are in the VPN?

   (c) How many bits are in the offset?

   (d) Again assume that the OS is using a *linear page table*. How many entries does this linear page table contain?

   Finally, the OS tracks the linear page table for a process by remembering it's *base address*, which we will assume for this problem to be the address where the page table is located in kernel physical memory. Given the address of the start of the page table (pt_base), and a VPN that you wish to translate into a PPN (physical page number), write some code to that **calculates a pointer to the right page table entry (pte)** for this VPN and returns it to the caller:

```
struct pte *p
find_pte(void *pt_base, int VPN)
{
  struct pte *p = ???
  return p;
}
```

   (a) Write your code here:

2. **Scheduling with Uncertainty.**

   Assume we have three jobs that enter a system and need to be scheduled. The first job that enters is called A, and it needs 10 seconds of CPU time. The second, which arrives just after A, is called B, and it needs 15 seconds of CPU time. The third, C, arrives just after B, and needs 10 seconds of CPU time.

   For all questions involving round-robin, assume that there is **no cost to context switching.** Also assume that if job X arrives just before Y, a round-robin scheduler will schedule X before Y.

   (a) Assuming a shortest-job-first (SJF) policy, at what time does B finish?

   (b) Assuming a longest-job-first (LJF) policy, at what time does B finish?

   (c) Assuming a round-robin policy (with a time slice length of 1 second), when does job A finish?

   (d) Assuming a round-robin policy (with a time-slice length of 1 second), when does job B finish?

   (e) Assuming a round-robin policy (with an **unknown time-slice which is some value less than or equal to 2 seconds**), when does job B finish?

   (f) Assuming a round-robin policy (with an **unknown time-slice**), for what values of the time-slice will B finish before C?

   Of course, SJF is unrealistic, because usually the OS doesn't know how long jobs are. In this system, though, the user gives the OS an estimate. The problem is that the users aren't so good at estimation. In fact, if they tell you a job will last N seconds, it might last anywhere between $N - 5$ and $N + 5$ seconds. But, being a nice, trusting OS, the OS assumes the user is exactly right.

   (a) Assuming SJF, what estimates (by the user) will lead the OS to make the worst decisions for these jobs in terms of achieving the lowest average response time?

   (b) In this case, what will the average response time of each job be?

   (c) What would the average response time of each job be if you instead had used LJF?

   (d) Assume we can arbitrarily change the run-time of job B. What should the run-time of B be changed to so that SJF delivers the best average response time to jobs A, B, and C, even if there are bad estimates?

3. **Translation station.**

Assume a 32-bit virtual address space and further assume that we are using paging. Also assume that the virtual address is chopped into a 20-bit virtual page number (VPN) and a 12-bit offset.

The TLB has the following contents in each entry: a 20-bit VPN, a 20-bit PPN, and an 8-bit PID field. This TLB only has four entries, and they look like this:

VPN	PPN	PID
00000	00FFF	00
00000	00AAB	01
00010	F000A	00
010FF	00ABC	01

Note all these numbers are in hex. Thus, each represents four bits (e.g., hex "F" is "1111", hex "A" is "1010", hex "7" is "0111", and so forth). That is why the 20-bit VPN and PPN are represented by five hex numbers each.

Now, for each of the following *virtual address*, say whether we have a TLB hit or a TLB miss. **IMPORTANT: If it is a hit, provide the resulting physical address (in hex).** Note: unless said otherwise, virtual addresses are also in hex.

(a) PID 00 generates the virtual address: 00000000

(b) PID 01 generates the virtual address: 00000000

(c) PID 00 generates the virtual address: FF00FFAA

(d) PID 00 generates the virtual address: 0010FFAA

(e) PID 01 generates the virtual address: 0010FFAA

(f) PID 00 generates the virtual address: 000000FF

(g) PID 01 generates the virtual address: 00000FAB

(h) PID 00 generates the virtual address: 010FFFFF

(i) PID 01 generates the binary virtual address 00000001000011111111010100001111

(j) PID 00 generates the binary virtual address 00000001000011111111010100001111

(k) PID 02 generates the virtual address: 00000000

4. **A Simple File System.** In this question, we are going to unearth the data and metadata from a very simple file system. The disk this file system is on has a fixed block size of 16 bytes (pretty small!) and there are only 20 blocks overall. A picture of this disk and the contents of each block is shown on the next page.

   The disk is formatted with a very simple file system, which looks a lot like that old Unix file system we have talked about in class. Specifically, the first block is a super block, the next 9 blocks each contain a single inode, and the final 10 blocks are data.

   The super block (block 0) has just four integers in it: 0, 1, 2, and 3, in that order.

   The root inode of this file system is in inode number 2 (at block 3 in the diagram).

   The format of an inode is also quite simple:

```
type:           0 means regular file, 1 means directory
size:           number of blocks in file (can be 0, 1, or 2)
direct pointer: pointer to first block of file (if there is one)
direct pointer: pointer to second block of file (if there is one)
```

   (assume that each of these fields takes up 4 bytes of a block)

   Finally, the format of a directory is also quite simple:

```
name of file
inode number of file
name of next file
inode number of next file
```

   (again assume that each field takes up 4 bytes of a block)

   Finally, assume that in all cases, no blocks are cached in memory. Thus, you always have to read from this disk all the blocks you need to satisfy a particular request. Also assume you **never** have to read the super block (just to make your life easier).

   That's it! Well, not quite; now you have to answer some questions:

   (a) To read the contents of the root directory, which blocks do you need to read?

   (b) Which files and directories are in the root directory? List the names of each file/directory as well as its type (e.g., file or directory).

   (c) Starting at the root, what are all the reachable regular files in this file system?

   (d) What are all the reachable directories?

   (e) What is the biggest file in the file system?

   (f) What are the contents of the biggest file?

   (g) What blocks are free in this file system? (that is, which inodes/data blocks are not in use?)

HINT: INODES (Block 1 – Block 9)

HINT: SUPER BLOCK (Block 0)

HINT: ROOT INODE (Block 3)

Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7	Block 8	Block 9
0	1	1	1	0	0	0	0	0	1
1	1	1	1	1	2	1	1	1	2
2	10	18	14	11	12	3	15	19	0
3	0	0	0	17	13	0	0	0	8

Block 10	Block 11	Block 12	Block 13	Block 14	Block 15	Block 16	Block 17	Block 18	Block 19
foo	3	7	hi	a	10	11	i	cs	0
3	4	8	10	0	goo	bar	luv	6	0
bar	5	9	you	b	11	oof	cs	537	0
4	6	10	12	1	goo	da	537	7	0