CS-537: Midterm Exam (Fall 2009)

The “Lost” OS: Building on the Popularity of Television

Please Read All Questions Carefully!

There are eleven (11) 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... (insert sinister music here)

Name and Student ID: ____

Grading Page

	Points	Total Possible
Q1		20
Q2		20
Q3		20
Q4		20
Q5		20
Q6		20
Total		120

News flash: An amazing operating system's source code has been lost! Let's call this operating system "LostOS". Your job, alas, is to help recover some aspects of what LostOS was doing.

Fortunately, you still have the LostOS binary, and can boot the kernel and even runs some jobs on it. Thus, you can sometimes empirically see how LostOS behaves. Further, when it is running, you can examine the state of memory to help figure out what LostOS is doing. You even have access to some old versions of LostOS, so you can learn how it has changed over time. You will even (occasionally) get some tips from the Old Gray Person (OGP), an old-time hacker who remembers bits and pieces about what LostOS does. Overall, you have a lot of information at your disposal in the relentless pursuit of recovering the hidden secrets of LostOS. Lucky you!

So, are you up for the challenge?[1]

- Yes: ____
- No: ____
- Unsure: ____

[1] Note that whatever your answer is, it still leads to the same result: you taking this exam.

1. **Lost Scheduling.**

   According to the Old Gray Person (OGP), an old version of LostOS used a **Round Robin** scheduling policy in order to decide which jobs to run, with a 100-millisecond timeslice.

   (a) What are the major strengths of the Round Robin approach?

   (b) What are its major weaknesses?

   The creators of LostOS eventually decided to modify this initial Round Robin, and in later versions changed the timeslice to 1 millisecond (instead of 100).

   (c) In what way did this change make the LostOS scheduler better? (if any)

   (d) In what way did this change make the LostOS scheduler worse? (if any)

   Finally, according to OGP, the original creators got sick of Round Robin and decided to replace the decision of which process runs next with a Random approach (i.e., one that picks the next process to run at random, from the set of ready processes).

   (e) How does the Random approach compare to traditional Round Robin? Is it better, worse, or quite similar?

2. **Lost Base Values.**

An old version of LostOS, so says Old Gray Person (OGP), used segmentation (not paging) to provide some form of primitive virtual memory. In this problem, we'll use that knowledge to discover some lost base and bounds values.

OGP remember a few things. First, the system only had 2 segments (segment 0 for code and a growing heap, segment 1 for a negatively-growing stack). OGP also recalls on this old version of LostOS, the virtual address space size was only 128 bytes, and there was only 1K of physical memory.

OGP has one set of traces from one old program. In particular, the traces tell you which virtual address (VA) was accessed (in both hex and decimal forms), and then whether or not the access was valid or not (and hence a segmentation violation). If valid, the physical address (in both hex and decimal) are reported. Oddly enough, programs in LostOS were allowed to keep running after memory-access violations, and thus we have a long trace that continues even after such a violation occurred.

Here is the trace:

```
Virtual Address Trace
  VA 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000003ec (decimal: 1004)
  VA 0x0000001d (decimal:   29) --> VALID in SEG0: 0x0000021d (decimal:  541)
  VA 0x00000057 (decimal:   80) --> SEGMENTATION VIOLATION (SEG1)
  VA 0x0000001e (decimal:   30) --> SEGMENTATION VIOLATION (SEG0)
  VA 0x00000058 (decimal:   88) --> VALID in SEG1: 0x000003d8 (decimal:  984)
  VA 0x00000061 (decimal:   97) --> VALID in SEG1: 0x000003e1 (decimal:  993)
  VA 0x00000035 (decimal:   53) --> SEGMENTATION VIOLATION (SEG0)
  VA 0x00000021 (decimal:   33) --> SEGMENTATION VIOLATION (SEG0)
  VA 0x00000064 (decimal:  100) --> VALID in SEG1: 0x000003e4 (decimal:  996)
  VA 0x0000003d (decimal:   61) --> SEGMENTATION VIOLATION (SEG0)
  VA 0x0000000c (decimal:   12) --> VALID in SEG0: 0x0000020c (decimal:  524)
  VA 0x00000005 (decimal:    5) --> VALID in SEG0: 0x00000205 (decimal:  517)
  VA 0x0000002f (decimal:   47) --> SEGMENTATION VIOLATION (SEG0)
```

Now, let's try to recover some values. **Note: you may not be able to perfectly recover all values; if so, provide the best answer you can given the information above.**

(a) From this trace, what was the **base register of segment 0** set to?

(b) From this trace, what was the **bounds register of segment 0** set to?

(c) From this trace, what was the **base register of segment 1** set to?

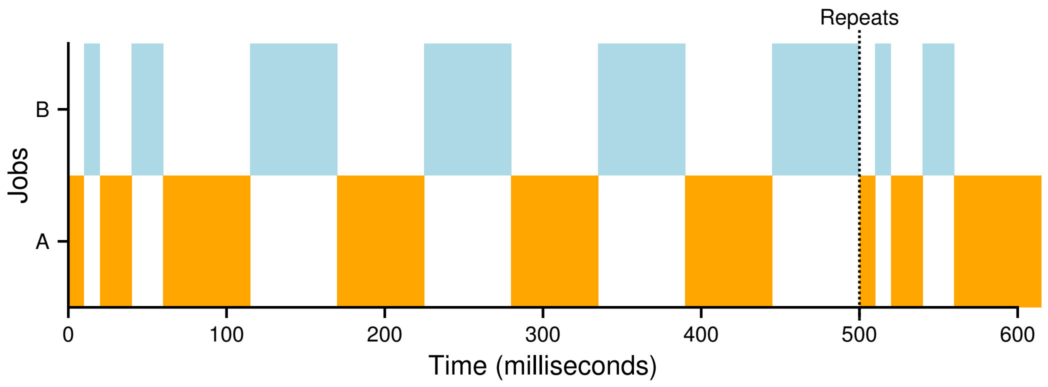
(d) From this trace, what was the **bounds register of segment 1** set to?

(e) In later versions of LostOS, OGP tells us that segmentation was dropped entirely and replaced with paging. What are some positives of the switch? What are some potential negatives? (i.e., why might you prefer segmentation over paging?)

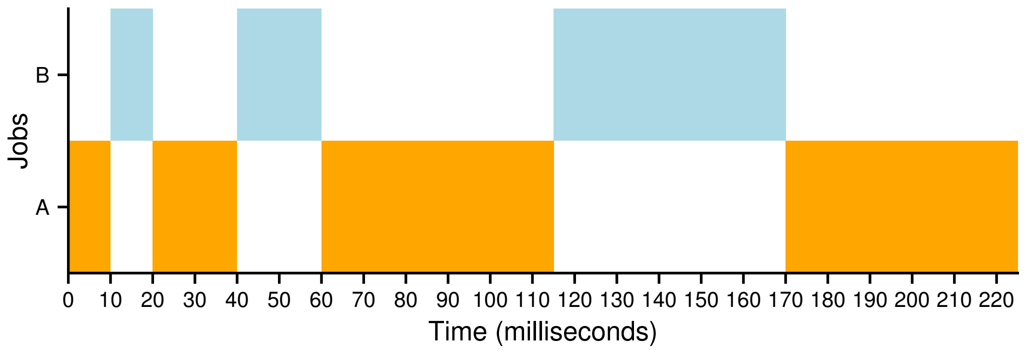
3. **Lost MLFQ Parameters.**

In the latest LostOS, there is an MLFQ (multi-level feedback queue) scheduler. Unfortunately, all of its parameters have been lost! Your job here is to examine a scheduling trace and determine what those critical parameters are.

Here is a timeline of what happens when two **CPU-bound** (no I/O) jobs, A and B, run:



This figure shows when A and B run over time; note that after 500 milliseconds, the behavior repeats, indefinitely (until the jobs are done). To help you further, a closeup of the first part of the graph is shown here:



Now, for some questions:

(a) How many queues do you think there are in this MLFQ scheduler?

(b) How long is the time slice at the top-most (high priority) queue?

(c) How long is the time slice at the bottom-most (low priority) queue?

(d) How often do processes get moved back to the topmost queue?

(e) Why does the scheduling policy MLFQ move processes to higher priority levels (i.e., the topmost queue) sometimes?

4. **Lost Translations**

An early version of the LostOS page replacement policy implemented a "perfect LRU" policy, with a little hardware support. The hardware support, according to the Old Gray Person (OGP), was as follows: on every memory reference to a particular VPN, the hardware would update a timestamp in the page-table entry (PTE) of that process's page table. In this version of the system, LostOS used a simple **linear page table**.

OGP tells you some more information to help you out with this: the original LostOS PTE contains only a **valid bit**, a **present bit**, the **timestamp** (32 bits, with the time in seconds since 1970), and the **page-frame number (PFN)** (30 bits). Thus, the entire PTE is 64 bits (or 8 bytes). Finally, OGP remembers that the timestamp field is used as a disk address when the page has been swapped to disk. Unfortunately, OGP can't remember what order these were stored in memory!

Fortunately, OGP did save some memory dumps of page tables, and is willing to share a few entries from them here. These are all in hex values, of course (OGP only speaks in hex).

The First Entries from the Page Table for Process 1:

```
C0 00 01 00 4A DC 70 04
80 00 00 00 00 00 10 00
7F FF FF FF FF FF FF FF
7F FF FF FF FF FF FF FF
...
```

The First Entries from the Page Table for Process 2:

```
C0 00 00 10 4A DC 90 CB
C0 00 00 11 4A DC 90 C0
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
...
```

Recall that hex F = binary 1111, E = 1110, D = 1101, C = 1100, B = 1011, A = 1010, 9 = 1001, 8 = 1000, and so forth.

OGP also remembers that when the valid bit was set to 0, the rest of the fields were filled with all 0's or all 1's (but not a mix of 0's and 1's).

Finally, OGP gives you can example of what the current time looks like, in seconds, in hex (isn't OGP nice?):

```
0x4ADCCD3F
(which is decimal 1255984438 seconds since January 1, 1970)
```

And now, time for some questions (next page):

(a) LostOS has a valid bit; in general, what is the valid bit used for?

(b) LostOS has a present bit; in general, what is the present bit used for?

(c) Given the information above, what do you think the likely structure of the page-table entry (PTE) is? (draw a picture, and show clearly which bits you think are used for which purposes)

(d) Assuming your PTE structure, what would happen when **process 1** accesses virtual page 0?

(e) Assuming your PTE structure, what would happen when **process 1** accesses virtual page 1?

(f) Assuming your PTE structure, what would happen when **process 1** accesses virtual page 2?

(g) Assuming your PTE structure, what would happen when **process 2** accesses virtual page 0?

(h) Assuming your PTE structure, what would happen when **process 2** accesses virtual page 1?

(i) Assuming your PTE structure, what would happen when **process 2** accesses virtual page 2?

5. **Lost Replacement Policies (Part II).**

   Note: You should read Question 4 before reading/doing this question.

   In this question, we try to uncover the some of the policies LostOS used to replace pages. As was stated above, early versions of LostOS implements "perfect" LRU replacement policy by putting timestamps in the page table of each process, and having the hardware update said timestamps when a page is referenced (i.e., read from or written to).

   (a) Given this hardware support, what work does the OS have to do to implement perfect LRU? What is bad about this approach?

   At some point, OGP decided to change the system to use a simpler policy that only approximates LRU. Thus, the first change is to the hardware, to replace the timestamp on each access with a single bit, called the **reference bit**. Upon each access to a page, the hardware sets a reference bit for that page to 1.

   The second change is to the OS to use this bit to make replacement decisions. Unfortunately, this code has been lost as well. Assume the following data structures (note: the PTE structure uses integers for simplicity of coding, but in actuality would pack the data much more carefully using bitwise operators):

```
// a page table entry (PTE)
// NOTE: integers used for simplicity
typedef struct __pte_t {
  int      valid;
  int      present;
  int      referenced;
  unsigned pfn;        // the translation
} pte_t;

// a linear page table
// NOTE: if a process exists, the entire table is allocated
typedef struct __page_table_t {
  pte_t    pagetable[PAGE_TABLE_ENTRIES];
} page_table_t;

// the array of all page tables in system
// NOTE: if process X does not exist, p[X] will be NULL;
page_table_t *p[MAX_PROCESSES];
```

Continued on next page...

(b) Write the code that picks a page to replace by finding one that has not recently been referenced. Things to consider: Where to start the search for unused pages? Should the reference bit ever be cleared? For simplicity, the code should simply set two integers (pointers to which are passed into the `replace()` routine): `process`, to the process whose page should be replaced, and `VPN`, to the virtual page number of the page to be replaced.

```
// any global variables?


void replace(int *process, int *VPN) {









    // these are the variables you update with the answer
    *process =
    *VPN =
}
```

6. **Lost Material.**

   In this final question, this exam asks you questions about other parts of the material we covered. This "lost" material could have been developed into fuller, harder questions, but you only have 2 hours, after all.

   Please circle "True" if the statement is true, and "False" if it is, well, false.

   (a) The main reason to have a multi-level page table is to speed up address translation. True False

   (b) The main reason to have a hardware TLB is to speed up address translation. True False

   (c) Using a multi-level page table increases TLB hit time. True False

   (d) As the amount of addressable physical memory grows, the size of each page-table entry (PTE) must grow as well. True False

   (e) As the size of the virtual address space grows, the amount of space occupied by a linear page table also grows. True False

   (f) In a 2-level multi-level page table, each page directory entry should have a valid bit. True False

   (g) In a 2-level multi-level page table, the size of the page directory increases as the virtual address space gets larger. True False

   (h) When running on a virtual machine monitor, the OS still thinks it is in charge of the physical placement of pages. True False

   (i) When running on a virtual machine monitor, TLB hits take longer. True False

   (j) When running on a virtual machine monitor, TLB misses take longer. True False