COS318 Midterm Exam
Princeton University
Fall, 2011
Instructors: Profs. Margaret Martonosi & Vivek Pai

(Total Time = 80 minutes)

Question	Score
1-3	/30
4	/25
5	/25
6	/10
7	/10
Total	/100

- **This exam is closed-book, closed-notes. 1 single-sided 8.5x11" sheet of notes is permitted.**
- **Calculators are ok but unneeded. Laptop or palmtop computers are not allowed.**
- **Show your work clearly in the spaces provided in order to get full or partial credit.**
- **Excessively long and/or vague answers are subject to point deductions.**
- **If you are unclear on the wording/assumptions of a problem, please state your assumptions explicitly and work it through.**

Honor Code (Please write out these words and then sign): I pledge my honor that I have not violated the Honor Code during this examination.

_____

_____

_____

_____

Name:_____

Short Answer:

1. (10 points) Consider a scenario involving 4 jobs to be scheduled. Assume that scheduling and context switch overhead takes 0 time. Specify a case where non-preemptive First-Come-First-Serve scheduling will have a better average response time than pre-emptive Round Robin. That is, give a set of job durations and an arrival order for which this is true. Compute the average response time for both scheduling options

Some common cases where this might happen are: when there are equal duration jobs and they all arrive at the same time; when jobs arrive in increasing order of their durations; and when the time quantum of the pre-emptive scheduler is very small resulting in the response time for all jobs by increasing at least the length of the shortest job.

Grading: Any set of 4 jobs that satisfies the response time requirements in the question, along with clearly specified arrival times, durations, response time computations and pre-emption time quantum value will get full credit. Otherwise, credit is reduced as follows until 4 or less points remain: arrival times missing or unclear (-2), durations missing or unclear (-2), response time computations missing, unclear, or incorrect (-3), time quantum missing or unclear (-2), too few or too many number of jobs (-1). If the set of jobs does not satisfy the average response time requirement, credit is limited to 4 points. Incorrect interpretations of what the schedulers do limits the credit to 4 points.

2. (10 points) Recalling x86 real mode from the project, how many ways are there to use segmented addressing to specify physical address 0x10000? Justify your answer.

Real mode addressing uses a 16-bit segment (call it say “seg”) and a 16-bit offset (call it say “offset”), and the referred address is seg * 16 + offset. The address in this case is 0x10000 which limits the offset values to those with last 4 bits 0x0. Then, the number of valid (seg, offset) combinations (with 16 bit seg and offset) is 0x1000.

Grading: The answer 0x1000 with correct justification gets full credit. If the answer is off by 1 (because of not considering bit widths of seg/offset), it gets 7 points. Correct explanations/justifications for x86 real mode addressing with mistakes in counting gets 5 points. Any other answer gets 3 points.

Name:_____

3. (10 points) Consider three threads executing code as shown:

thread 1:	**thread 2:**	**thread 3:**
lock(A)	lock(B)	lock(C)
lock(B)	lock(C)	lock(A)
do_something1()	do_something2()	do_something3()
unlock(B)	unlock(C)	unlock(A)
unlock(A)	unlock(B)	unlock(C)

Does deadlock ever occur? Justify your answer, and if it does occur, make a change that eliminates it.

Yes. Deadlock occurs when circular waiting happens. That is, thread 1 is holding lock A and waiting for lock B; thread 2 is holding lock B and waiting for lock C; thread 3 is holding lock C and waiting for lock A. To eliminate deadlock, one method is to enforce the order of locking, say in alphabetical order. Therefore, we can swap lock(A) and lock(C) in thread 3 to eliminate deadlock. Other possible solutions include using a mutex to ensure two locks are acquired atomically by each thread.

Rubric:

10 - right answer, clearly worded

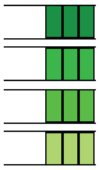
6-9 – right answer, minor error or unclarity exists

1-5 – wrong deadlock reason or incorrect deadlock elimination method

0- wrong answer

Name:_____

4. (20 points) Scheduling: In class, we discussed a pre-emptive scheduling technique with a prioritization approach involving 4 queues. Each queue has a different priority and a different corresponding time slice duration.



Priority	Time slices
4	1
3	2
2	4
1	8

Jobs start at the priority=4 queue (high priority but low time slice duration). If they use their full timeslice, then they are decremented to the priority 3 queue. Likewise, a priority 3 job that uses its full timeslice will get recategorized to priority 2 with an even longer time slice, and so on.

Priority 3 jobs only run if no priority 4 jobs are available (ie the priority 4 queue is empty). Likewise, priority 2 jobs only run if the priority 4 and 3 queues are empty, and so on.

a) (5 points) First, state why such an approach might be useful. Consider a range of jobs with varying durations, and with varying mixes of CPU and I/O requirements.

Most people got a and b correct. This approach separates different types of jobs into different queues. Long CPU-bound jobs will tend to use up their whole time quantum, so over time they will end up in the Priority=1 queue where jobs run at lower-priority but for longer timeslices. This allows short jobs, or long IO-bound jobs to run at higher priority (e.g. they will stay in the Priority=4 queue) for shorter bursts of time. Overall, this increases utilization and reduces response time for short jobs.

b) (7 points) Under what conditions might starvation occur for a job in this system?

Starvation can occur because if there is a steady stream of short jobs, or long IO-bound jobs that never use up their time quantum, there will always be priority=4,3,2 jobs to run. So any job that has the attributes (long CPU-intensive runtimes) to end up in the Priority=1 queue can stay there indefinitely while jobs from the other queues are serviced.

Name:_____

c) (13 points) Lottery scheduling techniques are appealing because they can reduce the likelihood of outright job starvation. Give the pseudo-code for a Lottery Scheduling approach that mimics the prioritization of the original scheme, but mitigates the starvation scenario from part b.

There were several different plausible solution approaches for this part. Some folks implemented a lottery approach to select a "winner" amongst the four queues, but then maintained FCFS behavior within the queues. This was fine and could get full credit as along as you properly handled the time durations of jobs in each queue.

Most students used a lottery approach across all the jobs (rather than across the four queues). This would also get full credit as long as you handled the details correctly. Namely, you needed to recognize priority changes and adjust ticket counts accordingly; show a selection approach for the schedule (even if just calling a random function GetWinningTicket()); adjust runtime intervals relative to priority.

A common error was to both increase and decrease ticket counts at the end of each scheduler interval, in a way that would cause the jobs to greatly diverge in priority (the rich get richer) after a few intervals.

-2: No clear treatment of adjusting time intervals
-3: Failure to adjust time intervals
-5: Both of the above. (Common)
-10: Some random ticketing scheme in place, but incomplete code structure
Other small point deductions for other errors.

Name:_____

5. (25 points) Hardware-Software Interface Issues. Your friend has just told you about a brand new processor with 256 general-purpose registers, and wants you to develop the OS for it. Using the mini-kernel from your projects as a model, please answer the following. Please be concise yet complete. For example, the following paragraph presents a reasonable analysis for non-OS issues: "Instruction encoding must change, either by eliminating three-operand instructions, or increasing the size of each instruction. This change stems from the fact that adding registers requires using more bits to specify each register. The code size is likely to increase due to either requiring more instructions or more memory to contain the same number of instructions. Instruction fetch bandwidth increases as a result."

a) (10 points) What parts of your system **will** be affected, how will they be affected, and why will they be affected this way?

The PCB/TCBs will grow larger to accommodate the extra space needed for the extra registers. The amount of space used by the kernel will increase, assuming the same number of processes. In the event that the kernel is not allocated much space, the maximum number of processes may be impacted.

The time to switch between user processes will increase, since all of the registers have to be saved/restored for the two processes involved.

b) (10 points) What parts of your system **may** be affected, under what circumstances might they be affected, and what effect this will have?

In all likelihood, the system call overhead will also grow, assuming that the kernel is saving/restoring all of the registers of the user process. This could also be placed in (a).

Interrupt cost is also likely to increase, unless the interrupt entry point tried to restrict register saving in order to reduce overhead.

If the cost of switching processes is large enough, it may have an impact on system efficiency that warrants increasing the scheduling quantum. Likewise, if the extra registers make the processes run more efficiently, it may make sense to reduce the quantum.

The cost of user-space thread switching may grow large enough that it's just not worth the overhead, and you abandon them in favor of kernel threads.

The process stack size may decrease as fewer variables need to be kept on the stack, so it may be the case that more processes can run. Conversely, one may be able to claim that the process stack size may increase, depending on the calling convention.

c) (5 points) Can you speed up any aspect of a system call using those registers? If so, state what can be changed and considerations in doing so.

Name:_____

Might as well pass parameters in the registers instead of using the stack.

It might also be that the kernel really doesn't need 256 registers, so the system call contract might specify that only a small subset of the registers have to be saved/restored by the caller.

Grading: on parts (a) and (b), 4 points, placed in whichever category, received full credit. Fewer responses received proportionately less credit. On part (c), the first answer received full credit, and the second answer alone received partial credit.

6. (10 points) Bootloader: Thinking back to Project 1, suppose that the instructions required to load the kernel in memory consume > 512 bytes of code. Propose a solution that will allow the bootloader to load the kernel, without changing anything about the BIOS.

Use a multi-stage bootloader. The BIOS loads the first stage from the disk's first sector, which contains code to load and jump to the larger second stage. The second stage loads the kernel into memory. All of the other normal bootloader considerations (self-relocation, knowledge of the kernel's location and size on disk, etc.) still apply.

Rubric:
10 - right answer, clearly worded
6-9 - right idea, incidental mistakes and/or lack of clarity
1-5 - misses the main idea but makes some valid point(s) specific to the boot process in this situation
0 - wrong or doesn't address the question

Name:_____

7. (10 points) Processes vs. Threads. Consider the following code sequence:

```
void
function1(void)
{
 static int a = 5;

 printf("%d\n", a);
 a++;
 printf("%d\n", a);
}

void
function2(void)
{
 int a    = 5;

 printf("%d\n", a);
 a++;
 printf("%d\n", a);
}


void
main(void)
{
< create processes/threads here >

 function1();
 function2();
 function1();
}
```

Show the output in the case where 2 processes are created, and in the case where 2 threads are created. In each case, assume that the processes/threads run sequentially.

Name:_____

Two processes

5
6
5
6
6
7
5
6
5
6
6
7
(sequence repeated once more if 3 processes)

Two threads

5
6
5
6
6
7
7
8
5
6
8
9
(if 3 threads, then add the following)
9
10
5
6
10
11

Partial credit variants:

The correct first half of each response was worth 2 points.
The correct second half of each response was worth 3 points.
Off-by-one errors lost two points total.