EECS 343: Midterm Exam *Sample*

From introductory concepts to virtual memory.

Fall 2016

Name (netid): *John Yossarian (jyo022)*

Some words of advice:

- Read all questions first.
- Start from the easiest one and leave the harder ones for later.
- Approximate results are almost always a valid answer; for sure I do not need 5-decimal precision answers!
- Write clearly; **if I can't read it, I can't grade it.**

Question	Total	Credited
1	10	
2	15	
3	20	
4	10	
5	15	
6	10	
7	20	
8	extra 15	
9	extra 15	
Total	100 (130)	

Good luck!

Problems

1. **(10 points)** Why is the process table needed in a timesharing system? Is it also needed in a computer systems in which only one process exists (taking over the entire machine until it is finished)?

   Answer: *The process table is used to keep, for every process in the system, all information that must be saved when the process is switched from running to ready or blocked state, so that it can be re-started later as if nothing had happend. Now, if there were only one process that is never context-switched out, then a process table would not be that useful.*

2. **(15 points)** When an interrupt or a system call transfers control to the operating system, a kernel stack area separate from the stack of the interrupted process is generally used. Why?

   Answer: *There are several reasons for using a separate stack for the kernel. Two of them are as follows. First, you do not want the operating system to crash because a poorly written user program does not allow for enough stack space. Second, if the kernel leaves stack data in a user program's memory space upon return from a system call, a malicious user might be able to use this data to find out information about other processes.*

3. **(20 points)** Consider the following preemptive priority-scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate α; when it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

   (a) What is the algorithm that result from $\beta > \alpha > 0$?

   (b) What is the algorithm that result from $\alpha < \beta < 0$?

   Answer: *Given that processes start with priority 0, the only interesting rates are additives (i.e. $p_i = p_{i-1} + \alpha$ and $p_i = p_{i-1} + \beta$). the first one is then FCFS and the second one is LCFS. You can easily see this with a simple example using $\alpha = 1$ and $\beta = 2$ and $\alpha = -2$ and $\beta = -1$ and processes that A, B and C arriving at times 2, 0 and 1 and with burst times of 3, 2 and 4, respectively.*

4. **(10 points)** Consider a page reference string for a process with a working set of size M, initially all empty. The page reference string is of length P with N distinct page numbers in it. For any page replacement algorithm,

   (a) What is a lower bound on the number of page faults?

   (b) What is an upper bound on the number of page faults?

   Answer:

   (a) *N*

   (b) *P*

5. **(15 points)** If FIFO page replacement is used with four page frames and eight pages, how many page faults will occur with the reference string 0172327103 if the four frames are initially empty? Now repeat this problem for LRU.

Answer: *FIFO 6, LRU 7*

6. **(10 points)** When segmentation and paging are both being used, as in MULTICS, first the segment descriptor must be looked up, then the page descriptor. Does the TLB also work this way, with two levels of lookup?

   Answer: *No. The search key uses both the segment number and the virtual page number, so the exact page can be found in a single match.*

7. **(20 points)** Consider a demand-paging system with the following time-measured utilization:

   - CPU utilization: 20%
   - Paging disk: 97.7%
   - Other I/O devices: 5%

   Which (if any) of the following will (probably) improve CPU utilization? Explain your answer:

   (a) Install a faster CPU
   (b) Get a bigger paging disk
   (c) Increase the degree of multiprogramming
   (d) Decrease the degree of multiprogramming
   (e) Install more main memory
   (f) Install a faster hard disk or multiple controllers with multiple hard disks
   (g) Add prepaging to the page fetch algorithm
   (h) Increase the page size

   Answer: *Clearly the system is thrashing. If the level of multiprogramming is reduced (d) or the amount of memory is increased (e), we could allocate more memory frames to the running processes, the system would page fault less frequently, the system would page fault less frequently and the CPU utilization would increase. A faster CPU (a) or bigger disk (b) or prepaging (g) would not do anything since the bottleneck would still be between the disk and main memory. Increasing the degree of multiprogramming (c) and increasing the page size (h) would only make things worse, since each process would have fewer frames available and the page fault rate would increase.*

8. **(Extra 15 points)** A straightforward way to implement a centralized lottery scheduler is to randomly select a winning ticket and then search a list of clients to locate the one holding that ticket. This requires a random number generation and $O(n)$ operations to traverse a client list of length n, accumulating a running ticket sum until it reaches the winning value. In [1] the authors suggest a simple optimization if the distribution of tickets to clients is uneven. Please explain their idea.

   Answer: *We mentioned this in class, just order clients in decreasing ticket count will reduce the average search length.*

9. **(Extra 15 points)** *Another paper-reading related question.*

References

[1] C. Waldspurger and W. Weihl, "Lottery scheduling: Flexible proportional-share resource management," *In Proc. of OSDI*, October 1994.