Innopolis University, Academic Year 2017/2018
First Retake of Operating Systems – 18th December 2017
Max 20 points + 6 points (~30%) extra credit – Total time: 90 minutes

1. (3 points) Why is the process table needed in a timesharing system? Is it also needed in personal computer systems running UNIX or Windows with a single user?
The process table is needed to store the state of a process that is currently suspended, either ready or blocked. Modern personal computer systems have dozens of processes running even when the user is doing nothing and no programs are open. They are checking for updates, loading email, and many other things, On a UNIX system, use the ps -a command to see them. On a Windows system, use the task manager.

2. (4 points) Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S , which is effectively wasted (overhead). For round-robin scheduling with quantum Q , give a formula for the CPU efficiency for each of the following: (a) $Q = \infty$, (b) $Q > T$, (c) $S < Q < T$, (d) $Q = S$, (e) Q nearly 0
The CPU efficiency is the useful CPU time divided by the total CPU time. When $Q \geq T$, the basic cycle is for the process to run for T and undergo a process switch for S. Thus, (a) and (b) have an efficiency of $T/(S + T)$. When the quantum is shorter than T, each run of T will require T/Q process switches, wasting a time ST/Q. The efficiency here is then

$$T / (T + ST/Q)$$

which reduces to $Q/(Q + S)$, which is the answer to (c). For (d), we just substitute Q for S and find that the efficiency is 50%. Finally, for (e), as $Q \rightarrow 0$ the efficiency goes to 0.

3. (4 points) The readers and writers problem can be formulated in several ways with regard to which category of processes can be started when. Carefully describe three different variations of the problem, each one favoring (or not favoring) some category of processes. For each variation, specify what happens when a reader or a writer becomes ready to access the database, and what happens when a process is finished.?
Variation 1: readers have priority. No writer may start when a reader is active. When a new reader appears, it may start immediately unless a writer is currently active. When a writer finishes, if readers are waiting, they are all started, regardless of the presence of waiting writers.
Variation 2: Writers have priority. No reader may start when a writer is waiting. When the last active process finishes, a writer is started, if there is one; otherwise, all the readers (if any) are started.
Variation 3: symmetric version. When a reader is active, new readers may start immediately. When a writer finishes, a new writer has priority, if one is waiting. In other words, once we have started reading, we keep reading until there are no readers left. Similarly, once we have started writing, all pending writers are allowed to run.

4. (3 points) We consider a program which has the two segments shown below consisting of instructions in segment 0, and read/write data in segment 1. Segment 0 has read/execute protection, and segment 1 has just read/write protection. The memory system is a demand paged virtual memory system with virtual addresses that have a 4-bit page number, and a 10-bit offset. The page tables and protection are as follows (all numbers in the table are in decimal):

For each of the following cases, either give the real (actual) memory address which results from dynamic address translation or identify the type of fault which occurs (either page or protection fault):

Segment 0		Segment 1	
Read/Execute		Read/Write	
Virtual Page #	Page frame #	Virtual Page #	Page frame #
0	2	0	On Disk
1	On Disk	1	14
2	11	2	9
3	5	3	6
4	On Disk	4	On Disk
5	On Disk	5	13
6	4	6	8
7	3	7	12

(a) Fetch from segment 1, page 1, offset 3,
(b) Store into segment 0, page 0, offset 16,
(c) Fetch from segment 1, page 4, offset 28,
(d) Jump to location in segment 1, page 3, offset 32

Results:

	Address	Fault?
(a)	(14, 3)	No (or 0xD3 or 1110 0011)
(b)	NA	Protection fault: Write to read/execute segment
(c)	NA	Page fault
(d)	NA	Protection fault: Jump to read/write segment

5. (3 points) Consider a file whose size varies between 4 KB and 4 MB during its lifetime. Which of the three allocation schemes (contiguous, linked and table/indexed) will be most appropriate?
Since the file size changes a lot, contiguous allocation will be inefficient requiring reallocation of disk space as the file grows in size and compaction of free blocks as the file shrinks in size. Both linked and table/indexed allocation will be efficient; between the two, table/indexed allocation will be more efficient for random-access scenarios.

6. (3 points) The performance of a file system depends upon the cache hit rate (fraction of blocks found in the cache). If it takes 1 msec to satisfy a request from the cache, but 40 msec to satisfy a request if a disk read is needed, give a formula for the mean time required to satisfy a request if the hit rate is h . Plot this function for values of h varying from 0 to 1.
The time needed is h + 40 x (1 – h). The plot is just a straight line.

7. (30% extra credit) A typical printed page of text contains 50 lines of 80 characters each. Imagine that a certain printer can print 6 pages per minute and that the time to write a character to the printer's output register is so short it can be ignored. Does it make sense to run this printer using interrupt-driven I/O if each character printed requires an interrupt that takes 50 μ sec all-in to service?
The printer prints 50 x 80 x 6 = 24, 000 characters/min, which is 400 characters/sec. Each character uses 50 μ sec of CPU time for the interrupt, so collectively in each second the interrupt overhead is 20 msec. Using interrupt-driven I/O, the remaining 980 msec of time is available for other work. In other words, the interrupt overhead costs only 2% of the CPU, which will hardly affect the running program at all.

This is the end of the questions and solutions of the exam.