DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 346: Operating Systems
Fall 2019
Theory Assignment 1
Due date: Saturday September 21st before 23:59

Total marks: 70

Q.1. Consider a computer system with a single-core processor. There are two processes to run in the system: P_1 and P_2. Process P_1 has a life cycle as follows: CPU burst time of 15 units, followed by I/O burst time of minimum 10 units, followed by CPU burst time of 10 units. Process P_2 has the following life cycle: CPU burst time of 10 units, followed by I/O burst time of minimum 5 units, followed by CPU burst time of 15 units. Now answer the following questions:

a) Considering a *single programmed* operating system, what is the minimal total time required to complete executions of the two processes? You should explain your answer with a diagram.
b) Now considering a *multiprogrammed* operating system, what is the minimal total time required to complete executions of the two processes? You should explain your answer with a diagram.
c) *Throughput* is defined as the number of processes (tasks) completed per unit time. Following this definition, calculate the throughputs for parts a) and b) above. How does multiprogramming affect throughput? Explain your answer.

Q.2. Suppose that a multiprogrammed system has a load of N processes with individual execution times of $t_1, t_2, \ldots, t_N$. Answer the following questions:

a) How would it be possible that the time to complete the N processes could be as small as: *maximum* $(t_1, t_2, \ldots, t_N)$?
b) How would it be possible that the total execution time, $T > t_1 + t_2 + \ldots + t_N$? In other words, what would cause the total execution time to exceed the sum of individual process execution times?

Q.3. Which of the following instructions should be privileged? Explain your answer. (i) Read the system clock; (ii) clear memory; (iii) turn off interrupts; (iv) switch from user to monitor mode; (v) issue a trap instruction; (vi) copy from one register to another.

Q.4. Answer the following questions:

a) Why need *system calls*? Explain from the following two perspectives: user's convenience and system protection.
b) Explain how a system call utilizes interrupts, user and monitor modes, and privileged instructions. What is achieved in return? Explain your understanding.

Q.5. Are interrupts always advantageous over polling? Explain your understanding with suitable example(s).

Q.6. Consider a preemptive operating system where processes have priorities and a running process gets preempted (i.e., forced to leave the CPU) as soon as a higher priority process is ready to run. The life cycle of a process, other than the very first process, begins with a "spawn" by another process and ends with either a regular "exit" by the process or a "terminate (process_id)" command by another process of equal or higher priority.

Each process is assigned an initial priority at spawn time and this priority remains unchanged during the entire life cycle. There are system resources, both hardware and software, for which a process can block if the resource is not free. A process can also be suspended by another process of equal or higher priority through the call "suspend (process_id)". A suspended process is resumed by the call "resume (process_id)". Note that a process can be in any state (i.e., running, blocked or ready) when suspended.

Processes communicate with one another via "send" and "receive" message passing primitives. "Receive" is always blocking, i.e., the calling process blocks if the message is not available. "Send" is always non-blocking. Illustrate the complete life-cycle of a process with the help of a *process state transition diagram*.

Q.7. When there is a context switch from one process to another, the OS kernel invokes the function *ContextSwitch* which saves the context of the currently executing process into its PCB and then inserts the process to an appropriate queue (i.e., ready queue or a blocked queue). It is necessary that *ContextSwitch* is atomic (i.e., unbreakable: either done or not-done; nothing in between). Explain the following:

a) Why must *ContextSwitch* be atomic?
b) Give an example scenario of what can go wrong if ContextSwitch is not atomic.
c) How can it be made atomic in practice?