Information and Computer Science Department
ICS 1431: Operating Systems

Second Semester 2019-2020 (192)
Homework & Programming Assignment No. 4
[Posted: Tuesday 24th March 2020]
[Due Date: Sunday 12th April 2020 @ 11:59 PM (Before Midnight)]

Submission Guidelines:

- Please answer if possible using your own words to reflect your understanding.
- You can past screen shots of executing commands or running processes.
- Submit an electronic copy to helmy@kfupm.edu.sa of your solution with the cover page includes:

HW_Prog_4_XXXXXXXXX_YourName_Lecture_Section_No

Where:

XXXXXXXXX is your 9 digit KFUPM ID.
YourFamilyName is your family name
Lecture_Section_No is the number of your ICS 431 lecture section

Submission should be made by hand to me during the class or in the office hours before the due date.

Important Notes:

- **Cheating is taken seriously**. Any cheating attempt will result in 0 for the HW or programming assignment.
- **EACH STUDENT IS REQUIRED TO DO THE HOMEWORK ALONE**. COPYING FROM ANY SOURCE IS REGARDED AS CHEATING.
- **No late submissions are allowed**.
- **Submissions via email can be accepted but you must submit printed hardcopies as well**.

A. The theoretical part of this assignment:

1. After reading the lectures' slides and the text book chapters 1, 2 and 3. Please try to answer the following exercises. **[36 Points, 3 Points for each exercise]**

- Chapter 6 Exercises No: 6.5, 6.7, 6.9. 6.12
- Chapter 7 Exercises No: 7.1, 7.5, 7.7, 7.12
- Chapter 8 Exercises No: 8.2, 8.4, 8.10, 8.14

B. The practical part of this assignment:

1. Solve the following programming problems in the text book, 7.17 in page 338 and 8.25 in page 387. **[24 Points]**

2. The goal of this programming assignment is to give you a basic introduction to the students IPC mechanisms and synchronization using semaphores. You need to write and run C programs (as processes on your Linux machine), and monitor their behavior. Consider the following problem: A program is to be written to print all numbers between 1 and 1000 (inclusive) that are not (evenly) divisible by either 2 or 3. This problem is to be solved using three processes (P0, P1, P2) and two one-integer buffers (B0 and B1) as follows:

    1. P0 is to generate the integers from 1 to 1000, and place them in B0 one at a time. After placing 1000 in the buffer, P0 places the sentinel 0 in the buffer, and terminates.
    2. P1 is to read successive integers from B0. If a value is not divisible by 2, the value is placed in B1. If the value is positive and divisible by 2, it is ignored. If the value is 0, 0 is placed in B1, and P1 terminates.
    3. P2 is to read successive integers from B1. If a value is not divisible by 3, it is printed. If the value is positive and divisible by 3, it is ignored. If the value is 0, P2 terminates.

Write a program to implement P0, P1, and P2 as separate processes and B0 and B1 as separate pieces of shared memory {each the size of just one integer. Use semaphores to coordinate processing. Access to B0 should be independent of access to B1; for example, P0 could be writing into B0 while either P1 was writing into B1 or P2 was reading. **[20 Points]**

3. The aim of this problem is to help you learn some of the CPU scheduling algorithm discussed in the class by implementing and simulating their performance. You need to program the following scheduling algorithms (Preferably in C): FCFS, SJF (with preemption) and Round Robin Scheduling. Each program will read from a file containing a list of processes with per-defined data for the process. The program will simulate the execution of the processes. It will print out the time taken by each process to complete (turnaround time) and the wait time and compute the average

turnaround time for all processes to execute as well as the standard deviation of that average.

Inputs: A filename from the keyboard read the file for the predefined data, and a possible time slice size (depending on which algorithm is used). The file containing the information on the processes will have each process on a separate line. The processes will be in the file in the order in which they arrive at the OS. Each line will have a process name that will be a string. Following that will be the arrival time of the process. The arrival time will be in reference to the previous process. Following this will be the total execution time. Next will be the elapsed time between I/O interrupts (system calls), next will be the time spent waiting and processing the I/O and finally the priority of the process as an integer (smaller values will have higher priority). It will look like this

P1	0	20.0	1.5	5.0	2
P2	2	15.0	2.0	6.0	1
P3	6	27.0	1.8	3.5	4
p4	4	36.0	2.1	2.6	3
x	x	x	x	x	x

Where, the xx indicates the end of the data.

Outputs: A prompt for which file is to be read. A prompt for the amount of time for a time-slice. A list of each process and the time it took for it to complete. Then an average time for processes to complete. Finally the standard deviation for the average time for processes to complete.

Example:

The name of the file to be read: < filename >

process name	turnaround time	total wait time
?	?	?

The standard deviation for the average process completion time was? For RR Scheduling: The time slice if required for your algorithm will be 3. You may assume that a swap (context switching time) is small enough that it can be ignored. Vary the time slice/quantum of RR scheduling from 1 to 10 sec (in steps of 1 sec) and plot a graph showing how the average turnaround time for processes vary with time slice/quantum. Also, plot a graph showing how the average waiting time for processes varies with time slice/quantum. **[20 Points]**