COMP 304- Operating Systems: Assignment 2

Due: April 23, midnight

Notes: This is an individual assignment. No late assignment will be accepted. Please submit your answers through blackboard. This assignment is worth 3% of your total grade. Corresponding TA: Mandana Bagheri-Marzijarani (mmarzijarani20@ku.edu.tr)

Problem 1

(25 pts) Consider the following table of arrival times, burst times and priorities for five processes P0 through P4. The context-switch time is 1 ms. Answer the parts A to D in your report.

Process	Arrival Time (ms)	Burst Time (ms)	Priority
P0	0	14	2
P1	0	8	1
P2	2	20	4
P3	4	4	3
P4	6	6	2

Part A

Draw four Gantt charts illustrating the execution of these processes using (A.1) First Come First Served, (A.2) Shortest Job First (SJF), (A.3) Non-preemptive with priority (lower priority number means higher priority) , and (A.4) Round-Robin (quantum = 6 ms) scheduling. Note that there is context switching overhead.

Part B

Calculate the waiting time of each process for each of the scheduling algorithms in Part A. Which of the schedules results in the minimal average waiting time?

Part C

Calculate the average turnaround time for each of the scheduling algorithms in Part A. Which algorithm has the lowest average turnaround time?

Part D

Based on the previous results, which one of the scheduling algorithms is the best in terms of response time?

Problem 2

(15+20+5 pts) For this question, use the code skeleton that is included in the assignment folder. In this code, you should complete the parts with a `TODO` comment and explain why you added/altered the code the way you did in the report. You should not change/add any code outside the explicitly mentioned `TODO` parts of the skeleton code both for Part A and B.

The provided skeleton code is a sample e-commerce application. The app has N items in `stock`. Every time the `sell()` function is called, one item is removed from stock and added to `sold` (Lines 35 to 50).

Part A: Racy Code

In this part, your goal is to fill in the `trigger_race_condition()` function so that when you run the code, more items are sold than available. We have added a `SLOWDOWN` macro in the middle of the critical section to increase the chances of context-switch happening at those points, and consequently, triggering a race-condition. The program will output a message if more stocks than available are sold.

The program creates new threads with the `create_new_thread()` function which executes the `sell()` method in a new thread, so that multiple instances of `sell()` run in parallel.

Note that you may need to run the program several times for the race-condition to be triggered. Once you observe a race condition, include a screen-shot as well as the explanations for the added code in the report.

To compile the code, use `gcc code.c -lpthread` which links the needed `pthread` library to the generated binary. You can then run the code via `./a.out`.

Part B: Fixing the race

Continuing from the code of Part A, you are required to ensure mutual exclusion to eliminate the race condition by using semaphores. APIs that create and use the semaphores are already provided in the skeleton code. Explain the changes in the code in your report.

Part C: Measuring time

Measure the time required to run the code for Part A and Part B separately, by adding a `time` command before program execution, e.g., `time ./a.out`. Report and explain the time differences.

Problem 3

(30pts) A well-known dentist has an office in Sariyer. As she got older, this dentist has a bad habit of taking several naps during day. Her office consists of a waiting room with N chairs and the treatment room can accommodate only one patient at a time. Here are some facts about this dental office.

- If there are no patients to be served, the dentist takes a nap.
- If a patient enters the dentist office and all chairs are occupied, then the patient leaves the office.
- If the dentist is busy with a patient but chairs are available, then the patient sits in one of the free chairs.
- If the dentist is asleep, the patient wakes her up.
- You can assume patients come in through one door and leave through the other. Only one patient can move at a time.

Write a monitor (in pseudocode) that coordinates the dentist and her patients. Represents dentist and each patient as separate processes. Explain the purpose of using each semaphore, mutex or condition variable that you include in your solution.

In your implementation, have the following monitor procedures:
get_dental_treatment: called by the patient, returns when treatment is done
get_next_patient: called by the dentist to serve a patient
finish_treatment: called by the dentist to let a patient out of the office

Problem 4

(5pts) Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Is this system is deadlock free? Show your work.