COMP 304- Operating Systems: Assignment 2

Due: 2 April 2018, 7.25 pm

Notes: This is an individual assignment. Any forms of cheating will be harshly punished! No late assignment will be accepted. Submit your answers through blackboard AND bring a hard copy to the exam (or drop it to the mailbox). This assignment is worth 3% of your total grade.
Corresponding TA: Najeeb Ahmad (nahmad16ku.edu.tr)

Problem 1

(10 pts) The processes are assumed to have arrived in the order of P1, P2, P3, P4 and P5 at time 0. Consider multi-level feedback queue scheduling where a process starts at first level and then if it is not finished in its quantum, it is moved from Queue i to the next level queue (Queue-$i+1$). Queue-1, which is the first level queue, is given a quantum of 4ms, Queue-2 is given a quantum of 8 ms, Queue-3 is given a quantum of 12 ms, and Queue-4 is scheduled as FCFS.

Process	Burst Time (ms)
P1	10
P2	20
P3	27
P4	18
P5	30

a) Draw the Gantt chart illustrating the execution of these processes. Calculate the average turnaround time. Assume no context-switch overhead.

b) Calculate the average turnaround time when the context switch overhead is 1 ms.

c) Compare multi-level queue scheduling with the round-robin scheduling using only the first level queue (queue-1) in terms of number of context switches. Which scheduling introduces less overhead? Show your work with by drawing the gantt chart for round-robin scheduling.

```c
#define MAX_LICENSES 5
int available_resources = MAX_LICENSES;

/* decrease available resources by count resources */
/* return 0 if sufficient resources available, otherwise return -1 */

int decrease_count(int count) {
  if (available_resources < count)
     return -1;
  else {
     available_resources -= count;
     return 0;
   }
}
/* increase available resources by count */
int increase_count(int count) {
     available_resources += count;
     return 0;
}
```

Problem 2

(6 pts) Many commercial software packages provide a given number of licenses, indicating the number of applications that may run concurrently. When the application is started, the license count is decremented. When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such requests will only be granted when an existing license holder terminates the application and a license is returned. The program segment is used to manage a finite number of instances of an available license.
The preceding program segment produces a race condition.
a. Identify the data involved in the race condition.
b. Identify the location (or locations) in the code where the race condition occurs.
c. Using a semaphore or mutex lock, fix the race condition.

Problem 3

(15 pts) A well-known dentist has an office in Sariyer but he has a bad habit of taking several naps during day. His office consists of a waiting room with N chairs and the treatment room can accommodate only one patient. 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 him 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 its patients. Represents dentist and each patient as separate threads. 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

(15 pts) In this question, you will learn the differences between fork, clone and pthread_create. The thread(s) created simply executes a hello world function. Consider the code segment listed below:

```
1  pid_t pid;
2
3  pid = create_process(); //1
4  if (pid == 0) {  /* Child process */
5      create_process(); //2
6      ...
7      create_thread(...);  //executes hello world only
8      ...
9  }
10 create_process();//3
```

a) Implement the above code using pthread_create and linux fork() system call to create threads and processes respectively and determine

- Number of unique processes created
- Number of unique threads created

b) Draw a process/thread tree and indicate processes as circles and the threads as rectangles.
c) Implement above code only using clone() system call for both creating threads and processes.

Problem 5

(4 pts) 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.