Indian Institute of Technology Guwahati
Department of Computer Science & Engineering

CS 558: Systems Lab
Assignment – 2

Instructions:

- Make sure that you read, understand, and follow these instructions carefully. Your cooperation will help to speed up the grading process.
- Following are generic instructions. Make sure that you also check carefully and follow any specific instructions associated with particular questions.
- In this assignment, you will explore the process scheduling, process synchronization and basic memory management in operating system.
- Complete the assignment with the best of your own capabilities. Create a single zipped/compressed file containing all your programs and related files.
- The compressed file must contain:
  - All your program files
  - Makefile to compile your programs
  - Readme file to explain your program flow
  - Proper code indentation and comments

  *** Each of the component mentioned above carries mark.**
- Please donot add any output files associated with the assignment in the compressed file.
- The file name should be as your roll no. Example: **184101001.zip.** When done, submit the file in Moodle on or before the submission deadline.
- Viva will be conducted for each group and marks will be alloted based on the individual performance.
- **Submission deadline: 11:55 PM, Tuesday 12th February, 2019 (Hard Deadline).**

Ethical Guidelines (lab policy):

- Deadlines: Deadlines should be strictly met. Assignments submitted after their respective deadlines will not be considered for evaluation.
- Cheating: You are expected to complete the assignment by yourself. Cases of unfair means and copying other's solution will not be tolerated, even if you make cosmetic changes to them. If we suspect any form of cheating, we are compelled to award -2x of the total mark alloted to the assignment.
- If you have issues in meeting the deadline, it is suggested that you consult the instructor and not use any unfair means to submit your assignment.

Question-1: Scheduling:

A. Consider a student Robert in M.Tech 1st year who has registered for the courses CS 123, CS 234, CS 345, CS 456 and CS 567. Each course has an assignment and the assignment has an announcement date, a submission date, and time (in hours) required to complete the assignment.

For example: Assignment-1 for course CS 123 is announced on 18-01-2019 and the submission date is on 21-01-2019. Therefore to complete the assignment, Robert has 4 days (96 hours). Similarly, the assignment for CS 234 is announced on 19/01/2019 and the submission date is also 19/01/2019. So Robert has exactly 24 hours to complete the assignment.

Robert can submit the assignment on or before the submission date. As the semester starts, assignments for different courses are announced. The first assignment announced is for course CS 123. In the meantime, the assignment for course CS 234 is announced on 19-01-2019 and similarly for CS 456, CS

345, and CS 567 as mentioned below. Robert can use a Time-Quantum (TQ) of **4-hours** per assignment. Within 4-hours of time-quantum he works on a particular assignment. When the time-quantum expires, he chooses the next assignment.

Course	**Announcement-Date**	**Submission-Date**	**Hours**
CS 123	18-01-2019	21-01-2019	59
CS 234	19-01-2019	19-01-2019	10
CS 345	20-01-2019	20-01-2019	09
CS 456	20-01-2019	21-01-2019	18
CS 567	21-01-2019	23-02-2019	24

In case he feels that scheduling one assignment is necessary over the others (if the deadline is closer), he gives that assignmnet a higher priority over the others so that he does not miss the submission deadline of any assignment.

Write a program in C to schedule the assignment as per the submission-deadline using appropriate scheduling algorithms.

Task 1: Print the Course number, Hours required for completion, Start time, Average waiting time, Average turnaround time, and Completion time (with date) for each course.

Task 2: Show the Gannt-chart (date-wise). For example:

18-01-2019	TQ – 1	TQ - 2	TQ – 3	TQ – 4	TQ – 5	TQ - 6
	1 4	**5 8**	**9 12**	**13 16**	**17 20**	**21 24**

*****In Readme file, mention the reason for choosing the scheduling algorithms used in this question.**

Question-2: Process Synchronization.

Consider a system with 3-coffee lover processes and one agent process. Each coffee lover continuously makes coffee and drinks. But to make coffee, the coffee lover needs 3 ingredients: coffee powder, sugar and milk. The agent has indefinite supply of all the three ingredients required to make the coffee. One of the coffee lover has coffee powder, another has sugar and the third has milk. The agent places two of the ingredients on the table. The process who has the remaining ingredient, makes the coffee and drinks, signaling the agent on completion. The agent then puts out another two of the three ingredients on the table and the cycle repeats.

Write a C program to code the agent and 3 coffee lovers as seperate processes. You shall achieve synchronization between them through semaphores.

Task 1: Input the number of iterations "N" that is to be used for making coffee.

Task 2: The agent should randomly choose two of the three ingredients required to make the coffee.

Task 3: Determine the number of semaphores required to achieve perfect synchronization among the coffee lovers. (Also, mention the reason in Readme file).

Task 4: For every iteration, print the ingredients that the agent placed on the table. Accordingly, print the coffee lover who have successfully prepared his coffee.

Question-3: Page allocation and Page replacement scheme.

Consider a multicore system that comprises of 3 cores. In each core, a process Pi ($1 \le i \le 3$) is running. The task is to calculate the sum of an array that consists of 1470 elements by the multicore system. [**Read the array elements from an input file.**]

The detailed explanation is mentioned below:

1) The array is divided into three equal halves and each process calculates the sum of the array elements within its specified range.

   For example, P1 calculates the sum of array elements from 0-489, P2 calculates from 490-979, and P3 calculates from 980-1469.
2) Initially, the array is stored in the secondary memory.
3) As the program starts, the array is brought from the secondary memory to main memory as pages. Thereafter, the processes start its execution.
4) Each page can hold only 40 array elements.
5) For the limited size of main memory, only 6 such pages can be brought at a time from secondary memory to main memory.

   This results in the usage of an optimal ***Page Allocation scheme*** as well as a ***Page Replacement scheme*** in the main memory.
6) After each process completes its execution, the total sum is calculated by the first process, P1 and displays the final output.
7) Write a C program to simulate the whole task with an optimal Page Allocation scheme as well as a Page Replacement scheme such that it results in:
   a) Less number of page faults.
   b) Less number of page replacements, and
   c) Less process waiting time.

****Assume that the main memory is initially empty.**

Kindly note that:

Each process is represented using a Proces Control Block (PCB) that contains:

a) **Process State**: Running, Waiting, Exit.
b) **Process ID**: 1, 2, or 3.
c) **Program Counter**: array index that is currently read.
d) **Memory Management Information** (that contains Page number currently brought into main memory).

Task 1: Whenever a process is changing its state, print the PCB of each process (in memory management information just print the page number brought by the respective process).

Task 2: Print the main memory contents whenever a new page is fetched into memory and pages are replaced.

Use this format:

$\mathbf{P}^{y1}_{x1}$	$\mathbf{P}^{y2}_{x2}$	$\mathbf{P}^{y3}_{x3}$	$\mathbf{P}^{y4}_{x4}$	$\mathbf{P}^{y5}_{x5}$	$\mathbf{P}^{y6}_{x6}$

Where x = #process number, y = #page-number fetched for the corresponding process, $\mathbf{P}_x$.

Task 3: Print the number of page faults and page replacements encountered by each process at the end of the simulation.