# Simulation Based Assignment Assessment Rubric

## Assessment Criteria

| Parameter | Weightage in % | Description |
|---|---|---|
| **Test Cases** | **20** | The code must satisfy the sample test cases i.e. for each set of input it must generate the desirable output. |
| **Concept Clarity** | **10** | The student need to specify the algorithms and the OS concepts they are applying on the given scenario based problem. |
| **Functional Requirements(boundary conditions, constraint satisfaction, etc)** | **50** | Code has to fulfill all the constraints and will satisfy the boundary conditions mentioned in the problem. It is mandatory to use C language. Solution should be implemented using OS concepts( System calls) |
| **Report submission** | **10** | The student has to submit the report as per the format specified. |
| **Use of GitHub Repository** | **10** | Student should upload the project on GitHub repository, Every week at least one revision should be done with a total of minimum 5 revisions during project lifecycle. Students uploading project in GitHub in the last week of submission would be subjected to DEDUCTION OF MARKS. |

**Note: Marks deduction on the basis of similarity index**

| **Plagiarism Weightage** | **Marks Deducted** |
|---|---|
| 50-60% | 03 |
| 60-70% | 06 |
| 70-80% | 09 |
| 80-90% | 12 |
| Above 90% | 27 |

## Report Format

Dear Students,

The individual project report template is built for letting us evaluate your individual understanding, capability and retention of the assigned project. While building answers to the questions from the template, you should be relating the project assigned to you and keep in mind the below three points and submit a write up to for the project.

**<u>Format of the report:</u>**
Text Size: 12
Text Style: Times New Roman
Line Spacing: 1.5 maximum

**<u>Mention the below in header of the word document</u>**
**Student Name:**
**Student ID**
**Email Address:**
**GitHub Link:**
**Code:** Mention solution code assigned to you

1. Explain the problem in terms of operating system concept? (Max 200 word)
**Description:**
2. Write the algorithm for proposed solution of the assigned problem.

**Algorithm:**
3. Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)
**Description (purpose of use):**
4. Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.
**Code snippet:**
5. If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.
**Description:**
6. Explain the boundary conditions of the implemented code.
**Description:**
7. Explain all the test cases applied on the solution of assigned problem**.**
**Description:**
**8. Have you made minimum 5 revisions of solution on GitHub?**
**GitHub Link:**

## Instructions

1. This assignment is a compulsory CA component.
2. The assignment is to be done on individual basis (no groups)
3. The assignment submission mode is **Online** only. Student has to upload the assignment on or before the last date on UMS only. No submission via e-mail or pen-drive or any media will be accepted.
4. Non-submission of assignment on UMS till the last date will result in **ZERO** marks.
5. The student is supposed to solve the assignment on his/her own. If it is discovered at any stage that the student has used unfair means like copying from peers or copy pasting the code taken from internet etc. **ZERO** marks will be awarded to the student.
6. The student who shares his assignment with other students (either in same section or different section) will also get **ZERO** marks.
7. If any student is not satisfied with his/her allotted question then he/she can approach me to get the question changed. In this case he/she will be allotted **Q8**(kindly go through the question before making a change request). But **only six** students can get their question changed. The decision to change the question will be done on **First Come First Serve basis**.

### Question Allotment Table

| Roll No. | Question No. | Roll No. | Question No. | Roll No. | Question No. |
|---|---|---|---|---|---|
| 1 | 6 | 22 | 6 | 43 | 6 |
| 2 | 7 | 23 | 7 | 44 | 5 |
| 3 | 4 | 24 | 4 | 45 | 4 |
| 4 | 3 | 25 | 3 | 46 | 7 |
| 5 | 2 | 26 | 2 | 47 | 2 |
| 6 | 7 | 27 | 1 | 48 | 1 |
| 7 | 6 | 28 | 6 | 65 | 6 |
| 8 | 5 | 29 | 5 | 50 | 3 |
| 9 | 4 | 30 | 4 | 51 | 5 |
| 10 | 3 | 31 | 3 | 52 | 4 |
| 11 | 2 | 32 | 2 | 53 | 7 |
| 12 | 7 | 33 | 7 | 54 | 2 |
| 13 | 6 | 34 | 6 | 55 | 1 |
| 14 | 5 | 35 | 5 | 56 | 6 |
| 15 | 4 | 36 | 4 | 59 | 3 |
| 16 | 3 | 37 | 3 | 60 | 4 |

| | | | | | |
|---|---|---|---|---|---|
| 17 | 2 | 38 | 2 | | |
| 18 | 1 | 39 | 1 | | |
| 19 | 2 | 40 | 5 | | |
| 20 | 3 | 41 | 6 | | |
| 21 | 4 | 42 | 2 | | |

## <u>Questions</u>

Q1. Write a multi-threaded C program that gives readers priority over writers concerning a shared (global) variable. Essentially, if any readers are waiting, then they have priority over writer threads -- writers can only write when there are no readers. This program should adhere to the following constraints:

Multiple readers/writers must be supported (5 of each is fine)

Readers must read the shared variable X number of times

Writers must write the shared variable X number of times

Readers must print:

The value read

The number of readers present when value is read

Writers must print:

The written value

The number of readers present were when value is written (should be 0)

Before a reader/writer attempts to access the shared variable it should wait some random amount of time

Note: This will help ensure that reads and writes do not occur all at once

Use pthreads, mutexes, and condition variables to synchronize access to the shared variable

.

Q2. Consider a scheduling approach which is non pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times. The priority can be computed as :

Priority = 1+ Waiting time / Estimated run time

Write a program to implement such an algorithm.

Q3. Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

Q4. Design a scheduling program to implements a Queue with two levels:

Level 1 : Fixed priority preemptive Scheduling

Level 2 : Round Robin Scheduling

For a Fixed priority preemptive Scheduling (Queue 1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2.

All the processes in second level queue will complete their execution according to round robin scheduling.

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.

2. Priority of Queue 2 has lower priority than in Queue 1.

Q5. Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only.  He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.

Q6. Write a program for multilevel queue scheduling algorithm. There must be three queues generated. There must be specific range of priority associated with every queue. Now prompt the user to enter number of processes along with their priority and burst time. Each process must occupy the respective queue with specific priority range according to its priority. Apply Round Robin algorithm with quantum time 4 on queue with highest priority range. Apply priority scheduling algorithm on the queue with medium range of priority and First come first serve algorithm on the queue with lowest range of priority. Each and every queue should get a quantum time of 10 seconds. CPU will keep on shifting between queues after every 10 seconds.

Q7. Create a Process Id (PID) manager that keeps track of free PIDs and ensures that no two active processes are having the same pid. Once a process terminates the PID manager may assigns its pid to new process.

Use the following constants to identify the range of possible pid values:

#define MIN PID 100

#define MAX PID 1000

You may use any data structure of your choice to represent the availability of process identifiers. One strategy is to adopt what Linux has done and use a bitmap in which a value of 0 at position i indicates that a process id of value i is available and a value of 1 indicates that the process id is currently in use.

Implement the following API for obtaining and releasing a pid:

• int allocate map(void)—Creates and initializes a data structure for representing pids; returns—1 if unsuccessful, 1 if successful

• int allocate pid(void)—Allocates and returns a pid; returns— 1 if unable to allocate a pid (all pids are in use)

• void release pid(int pid)—Releases a pid

Modify the above problem by writing a multithreaded program that tests your solution. You will create a number of threads—for example, 100—and each thread will request a pid, sleep for a random period of time, and then release the pid. (Sleeping for a random period of time approximates the typical pid usage in which a pid is assigned to a new process, the process executes and then terminates, and the pid is released on the process's termination.)

Q8. IndianRail has decided to improve its efficiency by automating not just its trains but also its passengers. Each passenger and each train is controlled by a thread. You have been hired to write synchronization functions that will guarantee orderly loading of trains. You must define a structure struct station, plus several functions described below.
When a train arrives in the station and has opened its doors, it invokes the function
station_load_train(struct station *station, int count)
where count indicates how many seats are available on the train. The function must not return until the train is satisfactorily loaded (all passengers are in their seats, and either the train is full or all waiting passengers have boarded).
When a passenger arrives in a station, it first invokes the function
station_wait_for_train(struct station *station)
This function must not return until a train is in the station (i.e., a call to station_load_train is in progress) and there are enough free seats on the train for this passenger to sit down. Once this function returns, the passenger robot will move the passenger on board the train and into a seat (you do not need to worry about how this mechanism works). Once the passenger is seated, it will call the function
station_on_board(struct station *station)
to let the train know that it's on board.
Create a file IndianRail.c that contains a declaration for struct station and defines the three functions above, plus the function station_init, which will be invoked to initialize the station object when IndianRail boots. In addition:

You must write your solution in C using locks and condition variables:

- lock_init (struct lock *lock)
- lock_acquire(struct lock *lock)
- lock_release(struct lock *lock)
- cond_init(struct condition *cond)
- cond_wait(struct condition *cond, struct lock *lock)
- cond_signal(struct condition *cond, struct lock *lock)
- cond_broadcast(struct condition *cond, struct lock *lock)

Use only these functions (e.g., no semaphores or other synchronization primitives).

- You may not use more than a single lock in each struct station.

- You may assume that there is never more than one train in the station at once, and that all trains (and all passengers) are going to the same destination (i.e. any passenger can board any train).
- Your code must allow multiple passengers to board simultaneously (it must be possible for several passengers to have called station_wait_for_train, and for that function to have returned for each of the passengers, before any of the passengers calls station_on_board).
- Your code must not result in busy-waiting.