CS558: Computer Systems Lab
Assignment Deadline: 11:59 pm, 18 April 2023

Instructions:

- This assignment is to be done in groups. Continue with the same group partner you had for the last assignment.
- The programs can be written in C/C++.
- Your code should have a readme file, a makefile, and it should be well commented. These will carry separate marks for each question.
- No extensions in submission are allowed. Delay in submission will lead to penalty inmarks.
- Assignments submitted before the deadline will only be considered for evaluation.
- Please do not email your assignments separately to the TAs, it will not be considered for evaluation.
- Your code will be checked for plagiarism. Any kind of academic dishonesty, plagiarism,etc. will lead to penalties.
- No sharing of code between students, submission of downloaded code is allowed.
- The first instance of code copying will result in ZERO marks for the assignment. The second instance of code copying will result in a `F' grade. Students may also be reported to the Students Disciplinary Committee, which can impose additional penalties.
- TOTAL MARKS = 20X3 (Excludes 10 marks for viva for each question, and includes 5 for readme, makefile, and, proper comments in the code)

1) Consider the following scenario. A town has a very popular restaurant. The restaurant can hold N diners. The number of people in the town who wish to eat at the restaurant, and are waiting outside its doors, is much larger than N. The restaurant runs its service in the following manner. Whenever it is ready for service, it opens its front door and waits for diners to come in. Once N diners enter, it closes its front door and proceeds to serve these diners. Once service finishes, the backdoor is opened and the diners are let out through the backdoor. Once all diners have exited, another batch of N diners is admitted again through the front door. This process continues indefinitely. The restaurant does not mind if the same diner is part of multiple batches.

   Write a C/C++ program to model the diners and the restaurant as threads in a multithreaded program. The threads must be synchronized as follows.

   - A diner cannot enter until the restaurant has opened its front door to let people in.
   - The restaurant cannot start service until N diners have come in.
   - The diners cannot exit until the back door is open.
   - The restaurant cannot close the backdoor and prepare for the next batch until all the diners of the previous batch have left.

2) Saraighat Bridge is the only bridge that connects the scenic north and south Guwahati and plays a vital role as it is one of the busiest bridges in the region. The bridge can become deadlocked if northbound and southbound people get on the bridge at the same time. (Let us assume the people are stubborn and are unable to back up.) Write a C /C++ code to implement the scenario of bridge using semaphores and/or mutex locks, that prevents deadlock.

   - Case 1 : Consider the scenario in which northbound people prevent southbound people from using the bridge, or vice versa.

   - Case 2 : Consider the scenario in which northbound people don't prevent southbound people from using the bridge, or vice versa.

   Represent northbound and southbound people as separate threads. Once a person is on the bridge, the associated thread will sleep for a random period of time, representing travelling across the bridge. Design your program so that you can create several threads representing the northbound and southbound persons.

3) You are a software developer for a major e-commerce platform that handles millions of transactions per day. Your platform runs on a distributed system consisting of multiple servers, and you are responsible for designing a process scheduling algorithm that can handle these transactions efficiently.

- The system consists of multiple servers, each running a different service. Each service handles a specific type of transaction. For example, one service handles payment processing, another handles order processing, and so on.
- Each service consists of a pool of worker threads, which are responsible for processing incoming requests. Each worker thread has a priority level assigned to it, and a certain amount of resources assigned to it. The priority level determines the order in which the threads are scheduled to process requests. The resources assigned to each thread determine the maximum number of requests it can handle simultaneously.
- Incoming requests are queued up and assigned to the appropriate service based on the type of transaction. Once a request is assigned to a service, it is further queued up and assigned to a worker thread based on its priority level.
- Your task is to design a process scheduling algorithm that can efficiently allocate resources to worker threads to process incoming requests. The algorithm should take into account the following factors:
  - The priority level of each worker thread
  - The number of available resources assigned to each worker thread
  - The type of transaction associated with each request
- Your algorithm should prioritize worker threads with higher priority levels and assign them resources accordingly. If a worker thread with a higher priority level is not available, the algorithm should assign resources to the next available worker thread with a lower priority level.
- The algorithm should also take into account the type of transaction associated with each request. Some types of transactions may require more resources than others. For example, payment processing may require more resources than order processing.

- Your program should read the input from the standard input and write the output to the standard output. The input will contain the following information:

  - The number of services **n** in the system
  - The number of worker threads **m** for each service
  - The priority level and resources assigned to each worker thread. Each worker thread should be on a separate line and its information should be separated by spaces in the following format: **priority_level resources**
  - The type of transaction associated with each request, and the number of resources required for that transaction. Each request should be on a separate line and its information should be separated by spaces in the following format: **transaction_type resources_required**
- Your program should output the following information:
  - The order in which requests were processed
  - The average waiting time for requests
  - The average turnaround time for requests
  - The number of requests that were rejected due to lack of resources
- Additionally, your program should output the following information during periods of high traffic:
  - The number of requests that were forced to wait due to lack of available resources
  - The number of requests that were blocked due to the absence of any available worker threads

----