SUMMARY Project#2

In this project we implemented multithreading and semaphores.
Multithreading- The ability of an OS or a program to run in multiple threads on a multiprocessor or uniprocessor is called multithreading. In simpler terms we can say managing more than one user at a time and managing multiple requests by a single user without needing to have multiple copies of the program running on the system. Since threads may interfere with each other while running in parallel we may have to deal with race condition (competition). We can have same resources needed at the same time by different threads. So we need mutual exclusion in this case or we can say that no two processes access the same resource at the same time. The piece of code that accesses a shared resource but that must not be concurrently accessed by any other thread of execution is called critical section of a code. Therefore we have to control the order of execution of our as well along with mutual exclusion. This can be dealt by Semaphores.

Semaphores- A semaphore is a variable or abstract data type that is used for controlling access, by multiple processes to a common resource in a concurrent system such as multiprogramming operating system or multithreading like in our case. In simpler words it is a thread synchronization construct that can be used send signals between threads for co-ordination and achieving mutual exclusion. A semaphore(s) is a variable which can be an integer value and may be initialized to a nonnegative number. The two operations - wait(s) this operation decrements the value and signal(s) this operation increments the value.

This project is about movie theatre simulation where we simulated a behavior of movie theatre in which we used threads to create a model between customers or consumers (which is buying a ticket for a movie along with other services) and provider (like agents selling tickets and workers like concession stand workers). Since a theatre can get multiple customers at time to buy tickets, to buy snacks, to enter theatre so we will be running multiple processes using multithreading where threads are coordinating with each other using semaphores and its two operation (wait and signal).

I simulated the above concept by using one thread per customer, one for box office agent, one for ticket taker and one for concession agent worker. I used three queues, one each for box office agent, ticket taker and concession stand worker along with semaphores for each of them to implement mutual exclusion. They all wait for customers to enter their respective queues which is handled by counting semaphores. A customer selects random movie (from the movie array) after which he/she enters the queue of box office agent. Initially we have taken two box office agents who serve the customers. Both the agents should be working in parallel using mutual exclusion. We also check for ticker availability using semaphore. After customer (let's say 1) buys a ticket he/she enters the ticket taker queue and waits there until ticket taker is free. Once finished [1] is signaled by the ticket taker the customer can randomly (50% chance) to choose whether he/she wants to go the concession stand. If not customer directly enters the theatre and If customer visit the stand, he/she can choose randomly what to eat and then can enter the movie theatre. Each place (ticker taker, concession stand and box office agent) has queue to wait and signal the customer where customer waits in queue and when signaled leaves for the next place. Execution ends when all the customers with ticket enter the theatre and their threads join

Difficulties in the project
There were many challenges like implementing mutual exclusion in threads (customer, ticker taker, concession stand worker and box office agent). Also I understood this later that a Queue is required for all the providers (agent, workers and ticket taker) since without maintaining queue there is a possibility that one customer may wait infinitely (starvation), also there will be no order if we don't maintain the queue. I had problem in mutual exclusion in most of the cases which was handled after I implemented queue for each provider. I also faced with a situation where only two of my threads were working and execution was infinite, this was a semaphore problem since I did not signal release for a customer in one of the threads.

This project was a great learning experience on how to implement multithreading and design semaphores in Java. Also I learned how a small mistake implementing semaphores and their in between communication can lead to whole program failure so they have to be carefully implemented.