**Project #2: Threads and Semaphores**

**Project purpose –**

The main purpose of this project is to understand how to work with semaphores and threads. While working with threads it’s very important to take care of synchronization when there is a possibility of accessing shared or global variables by multiple threads at the same time. The synchronization between threads is most crucial part which I implemented in this project using Semaphores. Semaphore allows threads to wait on particular part of the code to create synchronization. When semaphore value is 0, then it halts the thread till the value becomes one. Using semaphores, I successfully created mutual exclusion between threads. In this project, to simulate the post office I created a main class which initializes all the customer and postal worker threads. This class is responsible to join all the threads at the end. In this project, I learnt about following things –

1. Thread initialization
2. Use of semaphores to create mutual exclusion
3. Use of semaphores to synchronize the flow and communication between different threads.

**Implementation –**

I have implemented this project in Java. In this project, to simulate the postal office I created a main class as ‘PostOfficeSimulator’ which is responsible for following things –

* Global variables to exchange data between Customer and PostalWorker class.
* Initialization of all the semaphores required in the project.
* Method startPostalOffice() is responsible for below things –
  + Initializing the array of semaphores to identify and work based on particular customer.
  + Initializing all the Customer and PostalWorker threads.
  + Joining all the Customer and PostalWorker threads.

For simulating Customer thread, I created a class ‘Customer’ which is responsible for following things –

* When program runs, customerEntry semaphore will only allow 10 customer threads to enter into the program.
* Using mutex1-4 semaphores, I created a critical section in Customer class which is responsible for saving customerNumber and CustomerAction in global variables. This section is also responsible for saving the postalWorkerID from global variable to local variable. In this section it was necessary to create mutual exclusion which will allow only one thread at a time because global variables are shared across all the threads. If there is no mutual exclusion in this section, then one thread might change the value of another thread.
* Using postalWorkerAssigned, customerRequested and finished[custNum] semaphores, I synchronized the tasks such as requesting a task to buy stamps or mail a letter or mail a package between postalWorker and customer thread.

For simulating PostalWorker thread, I created a class ‘PostalWorker’ which is responsible for following things –

* When program runs, postalWorker semaphore will only allow 3 threads to enter into the program. Here we have initialized only three threads for postal worker, so this semaphore is doing no special work.
* In this class as well, Using mutex1-4 semaphores, I created a critical section in PostalWorker class which is responsible for saving postalWorkerID in global variable. This section is also responsible for saving the customerNumber and CustomerAction from global variables to local variables. In this section it was necessary to create mutual exclusion which will allow only one thread at a time because global variables are shared across all the threads. If there is no mutual exclusion in this section, then one thread might change the value of another thread.
* Using postalWorkerAssigned, customerRequested and finished[custNum] semaphores, I synchronized the tasks such as requesting a task to buy stamps or mail a letter or mail a package between postalWorker and customer thread.
* In function, workOnCustomerTask(), I have implemented the dummy logic to simulate three tasks such as buy stamps or mail a letter or mail a package. For buying stamps, I implemented a delay of 1000 ms and for mailing a letter, I implemented a delay of 1500 ms. For mailing a package, along with delay of 2000 ms, I implemented logic for creating mutual exclusion for scale resource which can be used by only one postal worker at a time. To create critical section I used ‘scale’ semaphore.

**Experience –**

This project was challenging and purely based on the concepts related to semaphores and threads. I learnt how to implement concurrency using semaphores. The experience was really good as now I feel really confident about all the concepts related to concurrency, threads and semaphores. When I implemented the project for the first time, somehow I messed it because I used many semaphores and later on I forgot which semaphore does which function. Then I decided that instead of wasting time in fixing the project, I implemented the whole project again. This time I went step by step to check the result after adding semaphores and checking if critical section is created or not. I was not sure if we need to join postal worker threads as well. Therefore I implemented it anyway. Overall, I love working on the assigned OS projects.