CS 149 Report
Team: Kernal
Team members: Jay Patel, Jagrut Desai, Shukan Shah and Hardik Joshi

Describe your software design.

We begin with defining the 75 students and enrolment duration of 120 quanta. Then we created struct for students that holds data such as student ID, priority ID, class ID, arrival time, decision, exit time etc. Each student has its own unique ID. The main function initialize the mutex for 3 different class sections and 3 different priority levels of students with GS, RS and EE. We have kept decision variable inside student struct to indicate whether student is able to enroll in the section or not. After that the main function starts to separate thread for 3 different class sections, a program thread and students thread. We treated queues in terms of arrays. Programmatically we control the operations of queues on corresponding array of GS, RS and EE.

After student threads arrive, we evaluate in which queue it will go first in depending on its priority. As we mentioned earlier, 3 threads are concurrently running, GS, RS, and EE; all three threads will take the first students from their respective queue. Now depending on Student's class section and class level, they will be placed in their respective class section. GS will sleep for the smallest time due to its highest priority and EE will sleep for the longest time due to its lowest priority. We keep 2 counters, 1 for how many students are there in each section and other is total count. If the class section counter reaches its threshold (in our case, 20) we start dropping the students from enrolling into that class. We extensively use Mutexes to protect our critical regions and synchronize the enrollment procedure. We also created a printing routine similar to "officehour.c" and kept a mutex for the printing event.

What data was shared and what were the critical regions?

We used enrollment counter for each class section. The total enrollment counter was shared among all three priority queues. For keeping counter for enrollment in each of the class section, we used pthread mutex. We also used the mutex for printing. Since events can occur from any one of the three threads of GS, RS, and EE as well as student arrives procedure, we needed to synchronize the printing routine. It avoids two threads from printing at the same time on the screen. Similarly, only one student gets enrolled at any given time. Thus, printing mutex was very vital.

What thread synchronization did you use?

Yes, we created program thread, student thread, priority thread and used pthread_join for synchronization. Synchronization is very crucial part of this program. In our assignment we used mutex on functions that handle queues and printing for proper synchronization. It gains the lock on data so that no other thread can interfere until the previous thread releases the lock.

This are the turnaround times we get for GS, RS and EE:
Turn around time for GS Queue is 1.71
Turn around time for RS Queue is 6.35
Turn around time for EE Queue is 8.60

For full report please refer to “output.txt” file.