OS Assignment 3 Design Document

Muzammil Tariq mt04305

November 22, 2020

1 The Way it Works

I used a total of 5 semaphores to apply synchronisation to the given problem. 1 semaphore called class_limit was used for restriction 1. 2 semaphores called os and pfun were used for restriction 2. 1 semaphore called TA_break was used for restriction 3. 1 last semaphore is used for mutual exclusion on the critical section. The flow of checking for this synchronisation is such that first it checks if the students from the other course are inside the office. Then it checks if the number of students inside the office is less than 3 or not. Then it checks how many students have been to the office since the last break. The reason I used semaphores is because they are easy to implement and can be used to implement both locks and condition variables.

Restriction 1: The semaphore called class_limit is initialised with the value MAX_SEATS. The function sem_wait() is called every time a student gets inside the office and the moment this value becomes 0 (which is when 3 students have entered the office) the next student to enter starts waiting outside the office. The function sem_post() is called on it every time a student leaves the office. So the moment the value becomes more than 0 another student can enter as there is room for 1 student now. The semaphore here is used as an ordering event as we could let at most 3 students inside the office at one instance.

Restriction 2: Both the semaphores os and pfun are initialised with the value 0. Reason for doing so is that if sem_wait() is called even one time on it the threads should stop right there. Every time a student tries to enter the office it checks if the amount of students of the other course in the office is more than 0. If the amount is 0 then it proceeds as usual otherwise it calls sem_wait() on the semaphore of that course (semaphore named os in case of an OS student entering the office and semaphore called pfun in case a PFUN student entering the office). While doing so it also counts the number of students(using an int variable called pfun_waiting for PFUN and another int variable called os_waiting for OS) waiting outside the office so when the amount of students of a course becomes 0 a for loop begins which calls sem_post() on the other course's semaphore os_waiting or pfun_waiting times. This relieves the other course student threads of their semaphore and lets them get inside the office. Both of these semaphores are used as condition variables here as the threads should stop the moment any thread from the other course gets inside the office.

Restriction 3: The semaphore called TA_break is initialised with the value TA_LIMIT. The function sem_wait() is called on it every time a student tries to enter a class. The moment the int variable students_since_break reaches the value TA_LIMIT(which is 10) and the int variable students_in_office reaches the value 0 (thus ensuring that students who entered the office have left) a for loop is started which calls sem_post() TA_limit times. This informs the student threads that the TA has taken his break. Before starting this for loop the int variable students_since_break is set to 0 thus telling the program that a new session is starting after the break and then the function called take_break() is called which sends the TA on a break. The semaphore here is used as an ordering event as we could let at most 10 students inside the office before the TA goes for a break.

Mutual Exclusion: The restrictions explained above follow mutual exclusion and maximise concurrency because of the way the flow is setup. The average real time it takes to run my implementation takes around 45 seconds on sample_input2.txt. Moreover, I have also used a semaphore called lock which acts as a lock to ensure mutual exclusion. The function sem_wait() is called on it when the thread is accessing the critical sections(where the values students_in_office, students_since_break, class_os_inoffice, and class_pfun are getting updated). The function sem_post() is called on it when the thread has left the criticial section. This semaphore is used as a lock over here as that is what was needed.

2 Removal of Synchronization Overhead

Restriction 1 and 3 were already in a state were synchronization overhead was at its least. In restriction 2 though I had used bools to make the conditions but I later realised that it could be done without them and that would reduce the complexity of the conditions to some extent. This was causing an issue where the maximum amount of students in the office would fall to 2 because of the way the synchronisation was done.