OSN Assignment 5

Shreyash Jain

2020101006

Report : Question 1 (without Bonus)

- This problem can be broadly broken down into 2 parts, one from the courses viewpoint and one from the standpoint of a student.
- So I create `num_students` threads which handle the routine of every spectator via the function `student_routine` which takes the index of array of student structure of its respective student as an argument.
- I also create `num_courses` threads which handle the routine of every course via the function `course_routine` which takes the index of array of course structure of its respective course as an argument.
- There are `num_labs` lab structures which act as a resource accessed by the course threads to allot TA and manage tutorials.
- **Note**: I have considered that the tutorial will only start if the tutorial has at least one seat filled. To prevent the TA from waiting endlessly for Tutorial Allocation, I stop the simulation when all students have exited by using `pthread_join` only on student threads.

A. Student Routine

This can be further broken down to following sections :

1. Filling Preference
2. Waiting for course allocation and tutorial to start
3. Opting for a course after tutorial
4. Move to next preference

Filling Preference

- For this, I make the thread corresponding to its student sleep for the time it takes them to fill their preference. Once, sleep finishes, we can assume that the preference has been filled. So I allot the current preference of the student as preference 1. I make use of the lock `students[id].slock` while changing the preferences or status of the student (`is_waiting_for_tut`, `is_waiting_for_opt`, etc.).

```
// Fill Preference
sleep(students[id].time_taken);
printf(ANSI_COLOR_BLUE "Student %d has filled in preferences for course registration\n" ANSI_COLOR_RESET, students[id].id);
pthread_mutex_lock(&students[id].slock);
students[id].current_preference_num = 1;
students[id].current_course = students[id].preference_1;
students[id].is_waiting_for_course_allocation = 1;
pthread_mutex_unlock(&students[id].slock);
```

Waiting for course allocation and tutorial to start

- Here, I make use of conditional variable `students[id].course_allocation_cond` to wait till the time I get a signal from the student's preferred course's course thread indicating that either the student has been allotted a seat in the course tutorial and the tutorial has ended or that the course has been withdrawn.

```
    pthread_mutex_lock(&students[id].slock);
    if (courses[students[id].current_course].is_withdrawn != 1)
    {
        // wait for course allocation and tut occuring
        pthread_cond_wait(&students[id].course_allocation_cond, &students[id].slock);
    }
```

- I set the status `is_waiting_for_opt = 1` if the student has attended the tut and is ready to make the decision whether to opt the course or not.
- I check if `is_waiting_for_opt` is 1, if it isn't that means that maybe the course is withdrawn while the student was waiting for a seat in the tut. If the case is so, I change his preference (using `move_to_next_preference(id)` function).

```
if (students[id].is_waiting_for_opt != 1)
  {
      if (courses[students[id].current_course].is_withdrawn == 1)
      {
          move_to_next_preference(id);
          if (students[id].out_of_options == 1)
          {
              pthread_mutex_unlock(&students[id].slock);
              goto student_exit;
          }
          pthread_mutex_unlock(&students[id].slock);
      }
  }
```

  - We can see here, if the student is out of options (no preference left), they leave the simulation.
  - We can see here, if the student is out of options (no preference left), they leave the simulation.
  - We can see here, if the student is out of options (no preference left), they leave the simulation.
- Now is `is_waiting_for_opt` is 1, that means the student is ready to make a decision. So I go to the next step.

Opting for a course after tutorial

- To make the decision, I generate a random number `probability` between 0 and 1 (inclusive)
- Now I compare this `probability` with the course opting probability i.e. Student Calibre * Course Interest. If `probability` is less than or equal to this product, that means the student has opted the course.

```
float probability = (float)rand() / (float)RAND_MAX;
if (probability <= students[id].calibre * courses[students[id].current_course].interest)
{
    // Chooses to opt
    students[id].course_opted = students[id].current_course;
    printf(ANSI_COLOR_BLUE "Student %d has selected course %s permanently\n" ANSI_COLOR_RESET, students[id].id, courses[students[id].c
    pthread_mutex_unlock(&students[id].slock);
    goto student_exit;
}
```

- Else, The student withdraws from this course and moves to next preference via the function `move_to_next_preference`

```
else
{
    // Chooses to move to next
    printf(ANSI_COLOR_BLUE "Student %d has withdrawn from course %s\n" ANSI_COLOR_RESET, students[id].id, courses[students[id].current
    move_to_next_preference(students[id].id);
    if (students[id].out_of_options == 1)
    {
        pthread_mutex_unlock(&students[id].slock);
        goto student_exit;
    }
}
```

Move to next preference

- Here I just update the `current_preference_num` , `current_course` and the `is_waiting_for_course_allocation` status for the student with given id in the students array.

```
if (students[id].current_preference_num == 1)
{
```

```
    students[id].current_preference_num = 2;
    students[id].is_waiting_for_course_allocation = 1;
    students[id].current_course = students[id].preference_2;
    printf(ANSI_COLOR_BLUE "Student %d has changed current preference from %s (priority %d) to %s (priority %d)\n" ANSI_COLOR_RESET, i
}
else if (students[id].current_preference_num == 2)
{
    students[id].current_preference_num = 3;
    students[id].is_waiting_for_course_allocation = 1;
    students[id].current_course = students[id].preference_3;
    printf(ANSI_COLOR_BLUE "Student %d has changed current preference from %s (priority %d) to %s (priority %d)\n" ANSI_COLOR_RESET, i
}
else
{
    students[id].current_preference_num = -1;
    students[id].current_course = -1;
    students[id].out_of_options = 1;
    students[id].has_exited = 1;
    students[id].is_waiting_for_course_allocation = -1;
    printf(ANSI_COLOR_BLUE "Student %d couldn't get any of their preferred courses\n" ANSI_COLOR_RESET, id);
}
```

All the above steps except filling preference are repeated till the student runs out of options or has permanently opted a course.

B. Course Routine

- Here, First I check if that specific course is withdrawn or not and updates its status. If the course gets withdrawn, I send a signal to the condition variable in students thread `course_allocation_cond` indicating course withdrawal.

```
int current_time = time_from_start();
int time_to_wait = g[i].time_since_start - current_time;
if (time_to_wait > 0)
    sleep(time_to_wait);
```

- I allot random number of seats between 1 and Max Seats for the course for the tutorial.
- After that I check is the course currently has a TA allotted, if not, I iterate through the lab Ids of the course and access TAs of those labs and check if a TA is available, I allot him the Course Tutorial.
- Then I allot the tutorial seats to the waiting students and fill the class till either all students are allotted a seat who have this course as current preference or the filled slots equal the total seats allocated for the tutorial.
- Then I start the tutorial, sleep for the tutorial duration and free the TA after it ends and signal all students that tut has ended by setting `is_waiting_for_opt = 1` and sending a signal to the conditional variable in students thread i.e. `course_allocation_cond`.
- Then I update the lab exhaustion status of all labs in this course after the TA has been utilized.
- **I repeat this process till course is withdrawn.**

Main function

- In the main function I take inputs, initialize threads, locks and condition variables.
- I create threads and join them.
- I just join the student threads as my program ends when all students have exited.