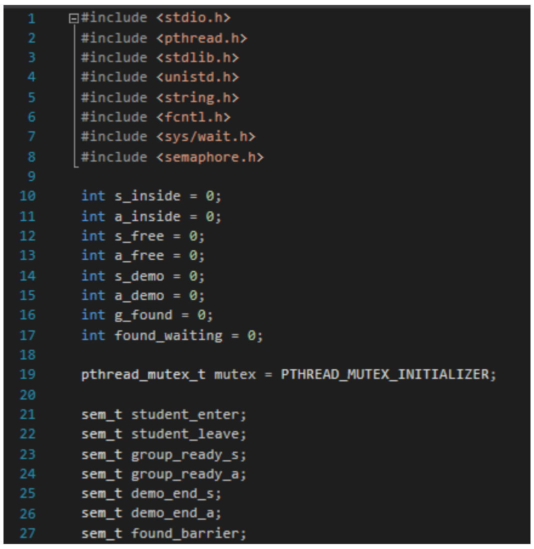
Programming Assignment (PA) – 3 (Demo Session Simulation)

CS307- Operating Systems

20 May 2023

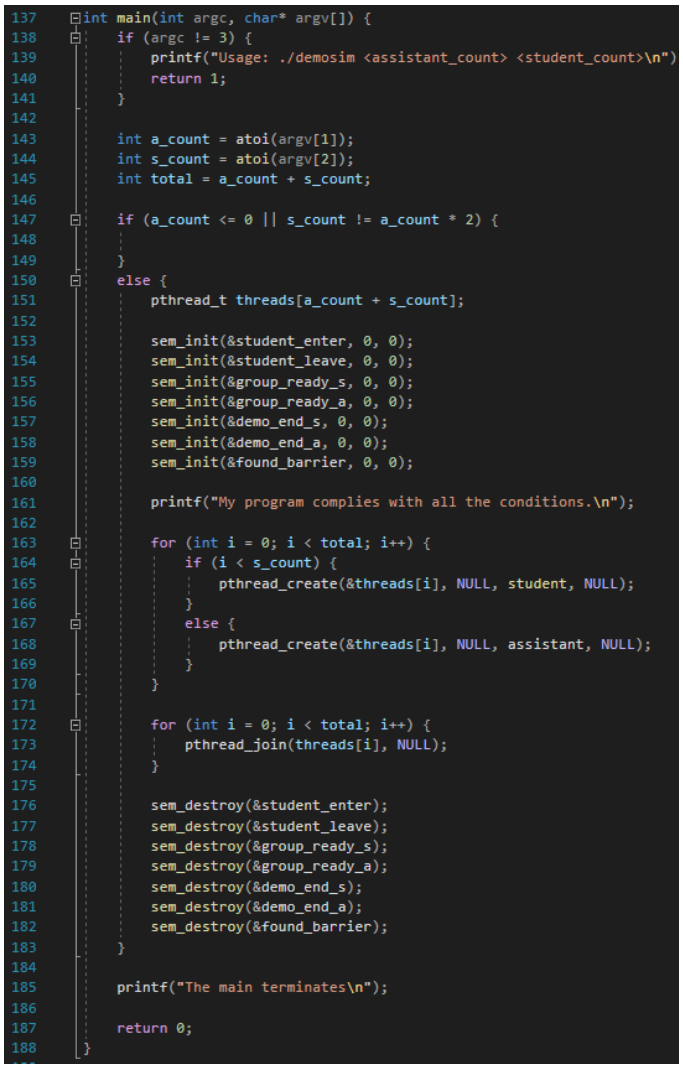
Solution Report

Ataollah Hosseinzadeh Fard
ID: 28610



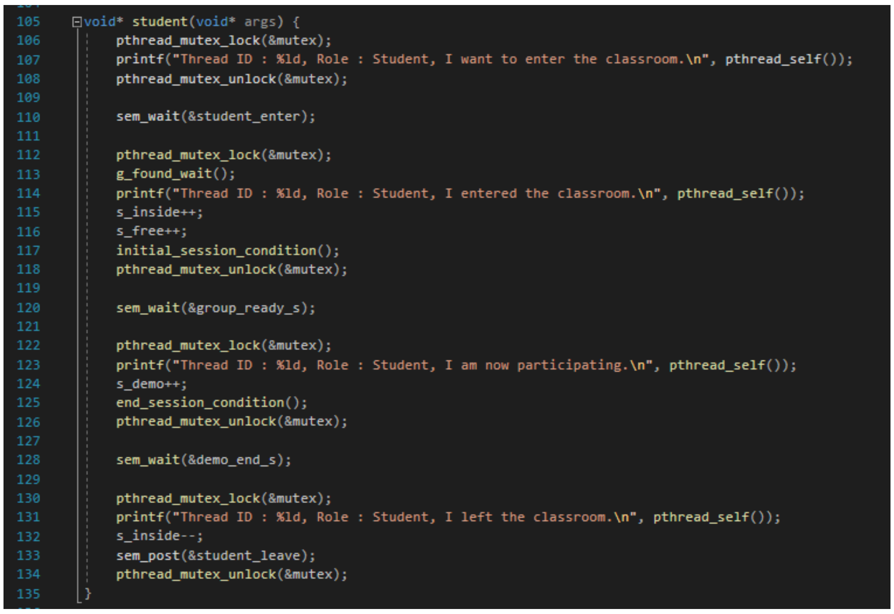
```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <semaphore.h>

int s_inside = 0;
int a_inside = 0;
int s_free = 0;
int a_free = 0;
int s_demo = 0;
int a_demo = 0;
int g_found = 0;
int found_waiting = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

sem_t student_enter;
sem_t student_leave;
sem_t group_ready_s;
sem_t group_ready_a;
sem_t demo_end_s;
sem_t demo_end_a;
sem_t found_barrier;
```

Added needed libraries also initialized needed mutex and counters, also defined needed semaphores and barrier to be used I program as global variables.



```c
int main(int argc, char* argv[]) {
    if (argc != 3) {
        printf("Usage: ./demosim <assistant_count> <student_count>\n")
        return 1;
    }

    int a_count = atoi(argv[1]);
    int s_count = atoi(argv[2]);
    int total = a_count + s_count;

    if (a_count <= 0 || s_count != a_count * 2) {

    }
    else {
        pthread_t threads[a_count + s_count];

        sem_init(&student_enter, 0, 0);
        sem_init(&student_leave, 0, 0);
        sem_init(&group_ready_s, 0, 0);
        sem_init(&group_ready_a, 0, 0);
        sem_init(&demo_end_s, 0, 0);
        sem_init(&demo_end_a, 0, 0);
        sem_init(&found_barrier, 0, 0);

        printf("My program complies with all the conditions.\n");

        for (int i = 0; i < total; i++) {
            if (i < s_count) {
                pthread_create(&threads[i], NULL, student, NULL);
            }
            else {
                pthread_create(&threads[i], NULL, assistant, NULL);
            }
        }

        for (int i = 0; i < total; i++) {
            pthread_join(threads[i], NULL);
        }

        sem_destroy(&student_enter);
        sem_destroy(&student_leave);
        sem_destroy(&group_ready_s);
        sem_destroy(&group_ready_a);
        sem_destroy(&demo_end_s);
        sem_destroy(&demo_end_a);
        sem_destroy(&found_barrier);
    }

    printf("The main terminates\n");

    return 0;
}
```

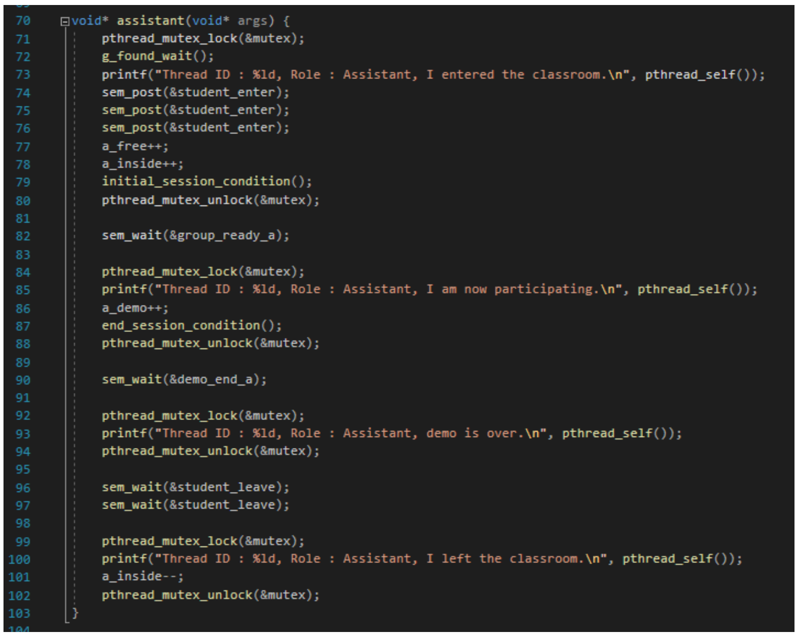
In main function, get assistant and student counts from arguments passed to main function if available, then checks counts, if assistant count is positive integer and students count is exactly 2 times of assistant count then program proceeds. initializes all mutexes and barrier with value of 0, creates needed assistant and student threads. Then waits for threads to finish, then deletes semaphores and barrier and main function terminates.



```c
void* student(void* args) {
    pthread_mutex_lock(&mutex);
    printf("Thread ID : %ld, Role : Student, I want to enter the classroom.\n", pthread_self());
    pthread_mutex_unlock(&mutex);

    sem_wait(&student_enter);

    pthread_mutex_lock(&mutex);
    g_found_wait();
    printf("Thread ID : %ld, Role : Student, I entered the classroom.\n", pthread_self());
    s_inside++;
    s_free++;
    initial_session_condition();
    pthread_mutex_unlock(&mutex);

    sem_wait(&group_ready_s);

    pthread_mutex_lock(&mutex);
    printf("Thread ID : %ld, Role : Student, I am now participating.\n", pthread_self());
    s_demo++;
    end_session_condition();
    pthread_mutex_unlock(&mutex);

    sem_wait(&demo_end_s);

    pthread_mutex_lock(&mutex);
    printf("Thread ID : %ld, Role : Student, I left the classroom.\n", pthread_self());
    s_inside--;
    sem_post(&student_leave);
    pthread_mutex_unlock(&mutex);
}
```

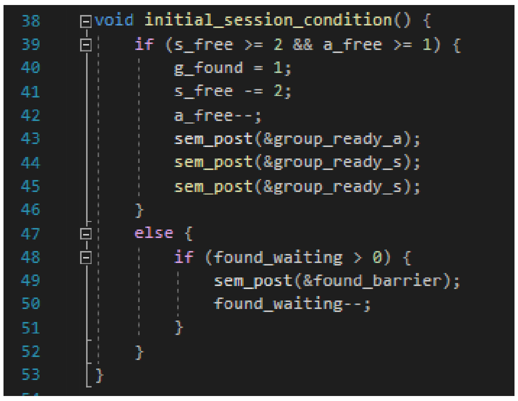
In student function, it first prints that, that students want to enter room. Then waits for “student_enter” semaphore to be signalled (released in other words), this semaphore will be posted by assistant. Then it will call “g_found_wait()”, which I will explain later. Then prints I am entered room and increases “s_inside” and “s_free” by 1. Then calls “initial_session_condition()”, which I will explain later. Then it waits for “group_ready_s” semaphore to be signalled. If it is signalled it means that a demo group with 2 students and 1 assistant has been assembled and then prints that it is participating and increases “s_demo” by 1. then calls “end_session_condition”, which I will explain later. After that it waits for “demo_end_s” semaphore to be signalled, if it is then it means that its demo session is over and it is allowed to print left the room and decrease “s_inside” by 1 and signals “student_leave” semaphore, this semaphore is being waited by assistant.

Important: At first, I had implemented the enter and leave of students and assistant using while loops but while I was in an office hour TA reminded me that infinite while loop may cause busy-wait problem, so I removed loops and used semaphores to keep enter and leave rules.



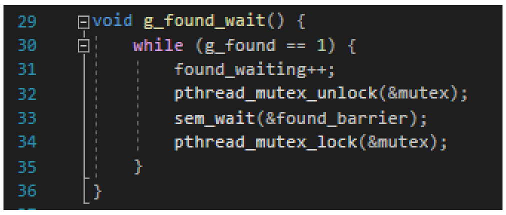
```c
void* assistant(void* args) {
    pthread_mutex_lock(&mutex);
    g_found_wait();
    printf("Thread ID : %ld, Role : Assistant, I entered the classroom.\n", pthread_self());
    sem_post(&student_enter);
    sem_post(&student_enter);
    sem_post(&student_enter);
    a_free++;
    a_inside++;
    initial_session_condition();
    pthread_mutex_unlock(&mutex);

    sem_wait(&group_ready_a);

    pthread_mutex_lock(&mutex);
    printf("Thread ID : %ld, Role : Assistant, I am now participating.\n", pthread_self());
    a_demo++;
    end_session_condition();
    pthread_mutex_unlock(&mutex);

    sem_wait(&demo_end_a);

    pthread_mutex_lock(&mutex);
    printf("Thread ID : %ld, Role : Assistant, demo is over.\n", pthread_self());
    pthread_mutex_unlock(&mutex);

    sem_wait(&student_leave);
    sem_wait(&student_leave);

    pthread_mutex_lock(&mutex);
    printf("Thread ID : %ld, Role : Assistant, I left the classroom.\n", pthread_self());
    a_inside--;
    pthread_mutex_unlock(&mutex);
}
```

In assistant function, it first calls “g_found_wait()” which I will explain later, then prints that it entered room and increases “a_free” and “a_inside” by 1. Also, signals “student_enter” 3 times which means tells 3 different students they can enter so this keeps following equation valid “s_inside < 3 * a_inside” so if equation is false then student waits for another assistant to enter and signal. then it waits for “group_ready_a” semaphore to be signalled. If it is signalled it means that a group has been assembled and it prints that it is participating and increases “a_demo” by 1. also calls “end_session_condition()” which I will explain later. Then it waits for “demo_end_a” semaphore to be signalled, if it is then it means that every person in session has started participating and now assistant can print demo is over. Then it waits for “student_leave” semaphore to be signalled twice, the reason it needs to be signalled twice is because each assistant can leave when 2 other students have left and signalled “student_leave” each of them. Then it prints it left room and decreases “a_inside” by 1 and function finishes.



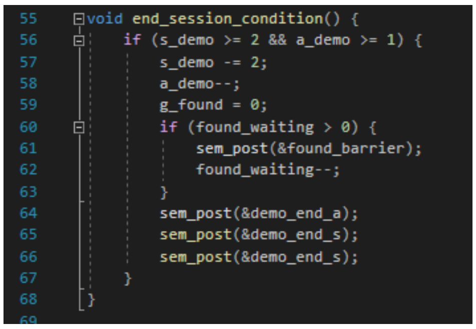
```
void initial_session_condition() {
    if (s_free >= 2 && a_free >= 1) {
        g_found = 1;
        s_free -= 2;
        a_free--;
        sem_post(&group_ready_a);
        sem_post(&group_ready_s);
        sem_post(&group_ready_s);
    }
    else {
        if (found_waiting > 0) {
            sem_post(&found_barrier);
            found_waiting--;
        }
    }
}
```

So, what “initial_session_conditions()” does is that it first checks if inside room is there any 2 or more students free and 1 or more assistants free to create a session, if yes then it makes “g_found” as 1, here this variable is integer but it will be used like a Boolean variable so it can be either 0 or 1 throughout the flow of program. Then since now 2 students and 1 assistant are no longer free, decreases “s_free” by 2 and decreases “a_free” by 1. then it signals “group_ready_a” once and “group_ready_s” twice, so this wakes 2 students and 1 assistant up and lets them know that they can start participating. However, if the numbers do not allow to form a session group then it goes to else part and checks if there is any person waiting for a previously formed team which means if “found_waiting” is larger than 1, then is decreases the “found_waiting” and signals “found_barrier” semaphore but this semaphore is used as a barrier, so as soon as a group formed and then second group want to be created this barrier will put new group to sleep until previous group has reached a point and I will explain that point later.



```
void g_found_wait() {
    while (g_found == 1) {
        found_waiting++;
        pthread_mutex_unlock(&mutex);
        sem_wait(&found_barrier);
        pthread_mutex_lock(&mutex);
    }
}
```

In “g_found_wait()” it creates a loop that will run until “g_found” is 1, this means that a group has been created and groups after them are sleeping. so inside loop it increases “found_waiting” by 1 and then waits for “found_barrier” to be signalled but since it must wait, it unlocks mutex first waits and then again lock mutex, this avoids any busy-wait because if it was inside mutex block it was never going to release since barrier has never signalled before and it causes a waiting until infinity.



```c
void end_session_condition() {
    if (s_demo >= 2 && a_demo >= 1) {
        s_demo -= 2;
        a_demo--;
        g_found = 0;
        if (found_waiting > 0) {
            sem_post(&found_barrier);
            found_waiting--;
        }
        sem_post(&demo_end_a);
        sem_post(&demo_end_s);
        sem_post(&demo_end_s);
    }
}
```

In “end_session_condition()” it tries to check if there are 2 or more students and 1 or more assistants participating in demo, if yes it means that all members of session have announced that they are participating so they are waiting to be able to leave or end demo. So, “s_demo” decreases by 2 and “a_demo” decreases by 1 also since this group has done “g_found” resets to 0 and if there is any group creating process waiting it wakes one of persons up and decreases “found_waiting” by 1. also signals “demo_end_a” once and “demo_end_s” twice. By these signals let’s 2 students to leave and 1 assistant to finish demo session.

<u>Important</u>: the reason I used “g_found” “found_waiting” and “found_barrier” is because at first, I had implemented program without those variable, counter and semaphore, and there was a 50% chance that program was messing order of group creation up when assistant count entered as input to program is larger than 1. For example, I will show and explain this problem in one of test runs for 2 assistants and 4 students. Lets assume assistant 1 and 2 students have entered room and since the “initial_session_condition()” can create a group, one of the members in this group starts participating, then another student enters room and since “initial_session_condition()” had signalled “group_ready_s” twice previously this 3rd student can start participating although it has entered after one of the students was started participating, so this mixes all group members with each other but after adding those 3 new variable, counter and barrier, it will make all new entered person to wait until currently created group has done and they are ready to leave room.

* Therefore, to sum up, there is 2 important aspects, first is that instead of using infinite loops for checking conditions to allow students to enter and assistants to leave I used semaphores which will not cause any busy-wait problems. Secondly, I used a complex waiting mechanism so that each group is assembled all other possible groups must wait until all group members of currently assembled group have participated, this makes sure that if there are multiple groups, they will not mix into each other.