TEACHING ASSISTANT PROBLEM

TEAM MEMBERS

Ch Mohith	AP21110010611
Rushendra Sai	AP21110010612
Jahnavi S	AP21110010613

Group -

Mentored by – Dr. **Krishna Siva Prasad Mudigonda**

TITLE – TEACHING ASSISTANT PROBLEM

- Let's look at the problem statement:
- **The Sleeping Teaching Assistant:**
- A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds the TA currently helping another student, the student "x" sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back at a later time. Using POSIX threads, mutex locks, and semaphores, implement a solution that coordinates the activities of the TA and the students.

WORKING RULE

Each will run as a separate thread. The TA will run as a separate thread as well. Student threads will seek help from the TA. If the TA is available, they will obtain help. Otherwise, they will either sit on a chair in the hallway or, if no chairs are available, will seek help at a later time (random period of time). If a student arrives and notices that the TA is sleeping, the student must notify the TA using a semaphore. When the TA finishes helping a student, the TA must check to see if there are students waiting for help in the hallway. If so, the TA must help each of these students in turn. If no students are present, the TA may return to napping. Simulating the TA providing help to a student is to have the appropriate threads sleep for a random period of time

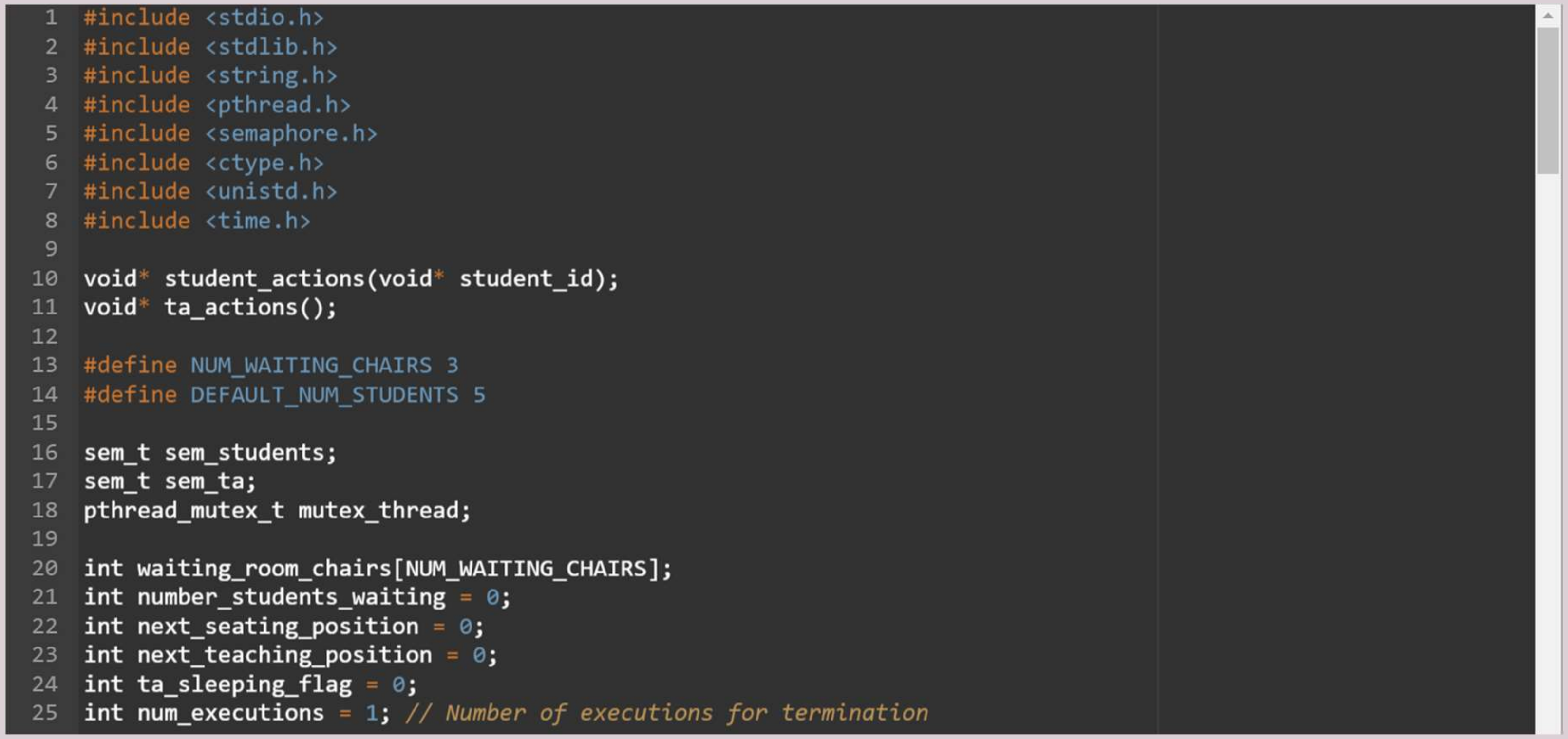
ALGORITHM OF TA PROBLEM

- Step 1: MAIN START
- Step 2: Get user input (Number of students)
- Step 3: Create TA thread (TA is sleeping)
- Step 4: Create student threads
- Step 5: MAIN END

INTEGERS USED

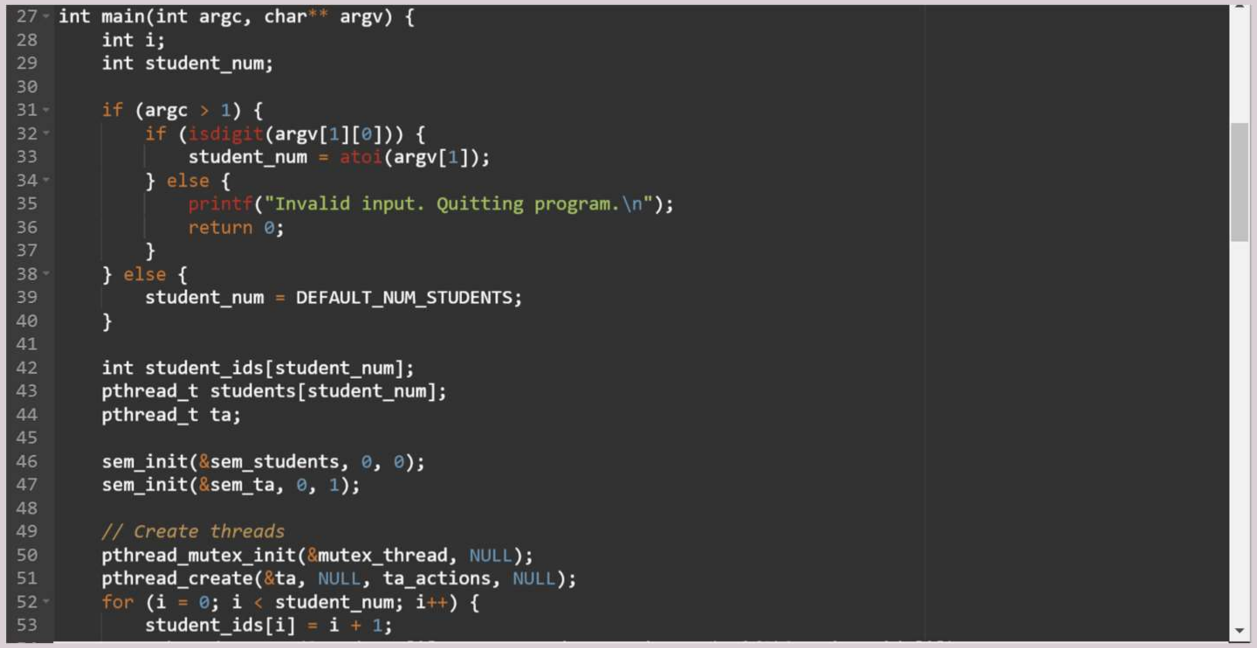
```
int waiting_room_chairs[NUM_WAITING_CHAIRS];
int number_students_waiting = 0;
int next_seating_position = 0;
int next_teaching_position = 0;
int ta_sleeping_flag = 0;
int num_executions = 1;
```

MAIN FUNCTION CODE



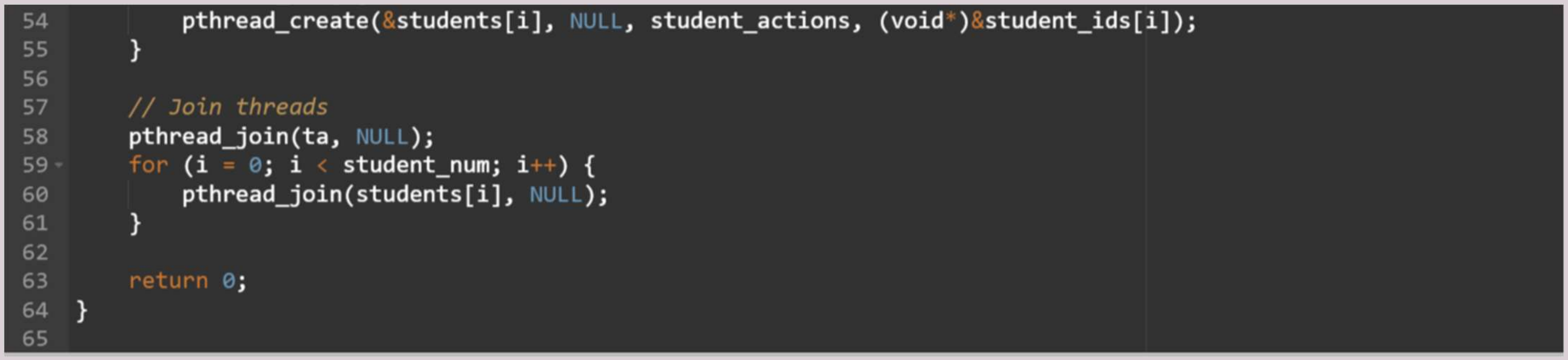
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <ctype.h>
#include <unistd.h>
#include <time.h>

void* student_actions(void* student_id);
void* ta_actions();

#define NUM_WAITING_CHAIRS 3
#define DEFAULT_NUM_STUDENTS 5

sem_t sem_students;
sem_t sem_ta;
pthread_mutex_t mutex_thread;

int waiting_room_chairs[NUM_WAITING_CHAIRS];
int number_students_waiting = 0;
int next_seating_position = 0;
int next_teaching_position = 0;
int ta_sleeping_flag = 0;
int num_executions = 1; // Number of executions for termination
```

MAIN FUNCTION CODE



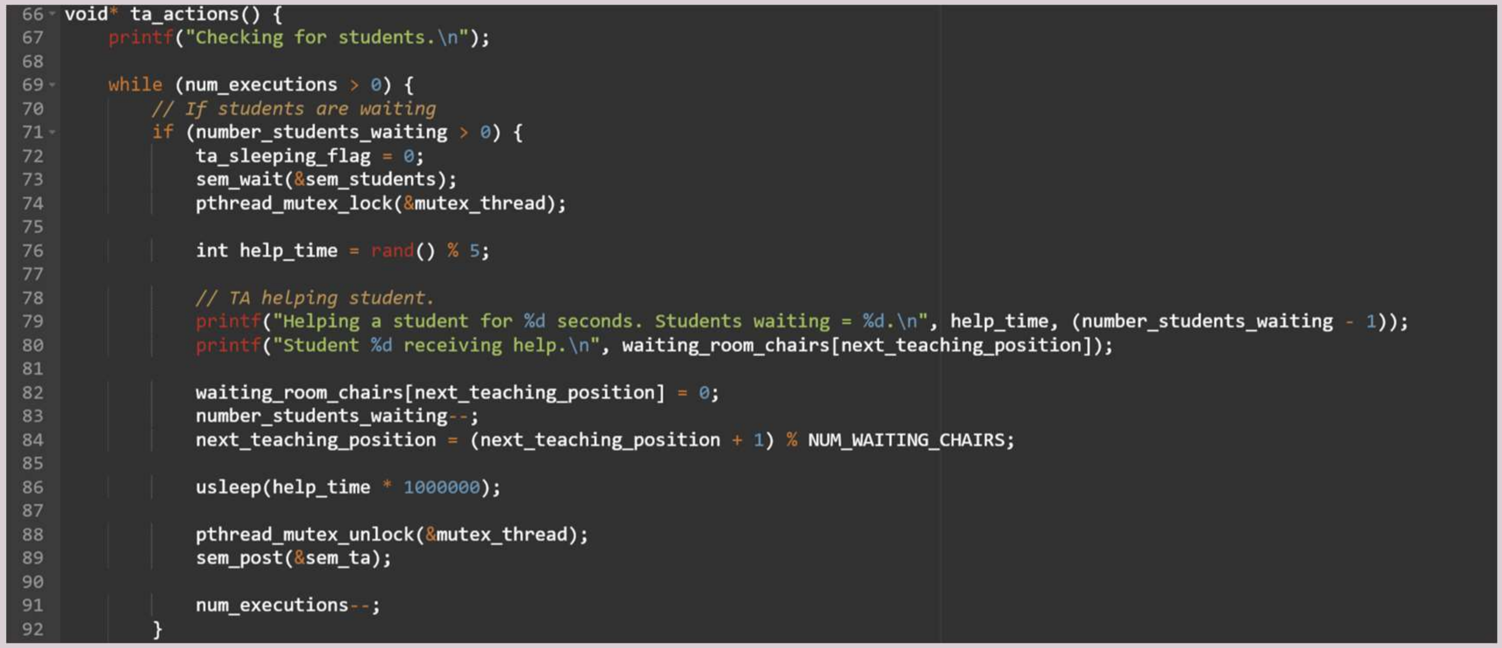
```c
int main(int argc, char** argv) {
    int i;
    int student_num;

    if (argc > 1) {
        if (isdigit(argv[1][0])) {
            student_num = atoi(argv[1]);
        } else {
            printf("Invalid input. Quitting program.\n");
            return 0;
        }
    } else {
        student_num = DEFAULT_NUM_STUDENTS;
    }

    int student_ids[student_num];
    pthread_t students[student_num];
    pthread_t ta;

    sem_init(&sem_students, 0, 0);
    sem_init(&sem_ta, 0, 1);

    // Create threads
    pthread_mutex_init(&mutex_thread, NULL);
    pthread_create(&ta, NULL, ta_actions, NULL);
    for (i = 0; i < student_num; i++) {
        student_ids[i] = i + 1;
```

MAIN FUNCTION CODE



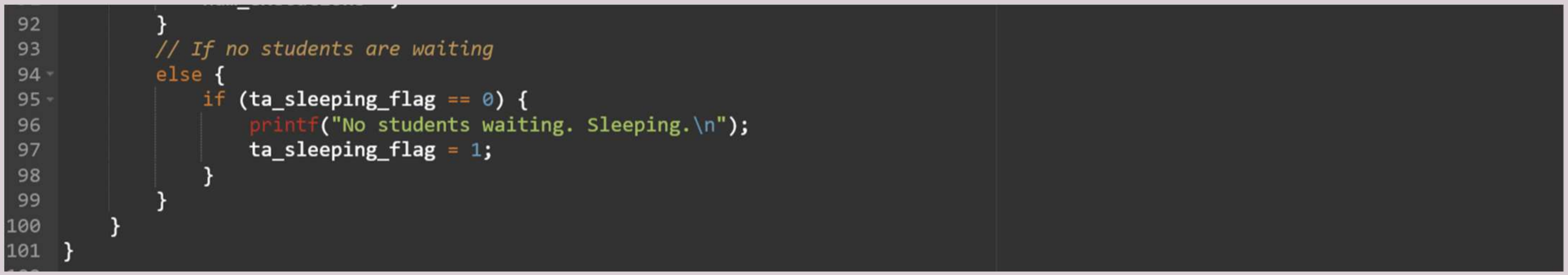
```
        pthread_create(&students[i], NULL, student_actions, (void*)&student_ids[i]);
    }

    // Join threads
    pthread_join(ta, NULL);
    for (i = 0; i < student_num; i++) {
        pthread_join(students[i], NULL);
    }

    return 0;
}
```

VOID TEACHING ASSISTANT FUNCTION CODE



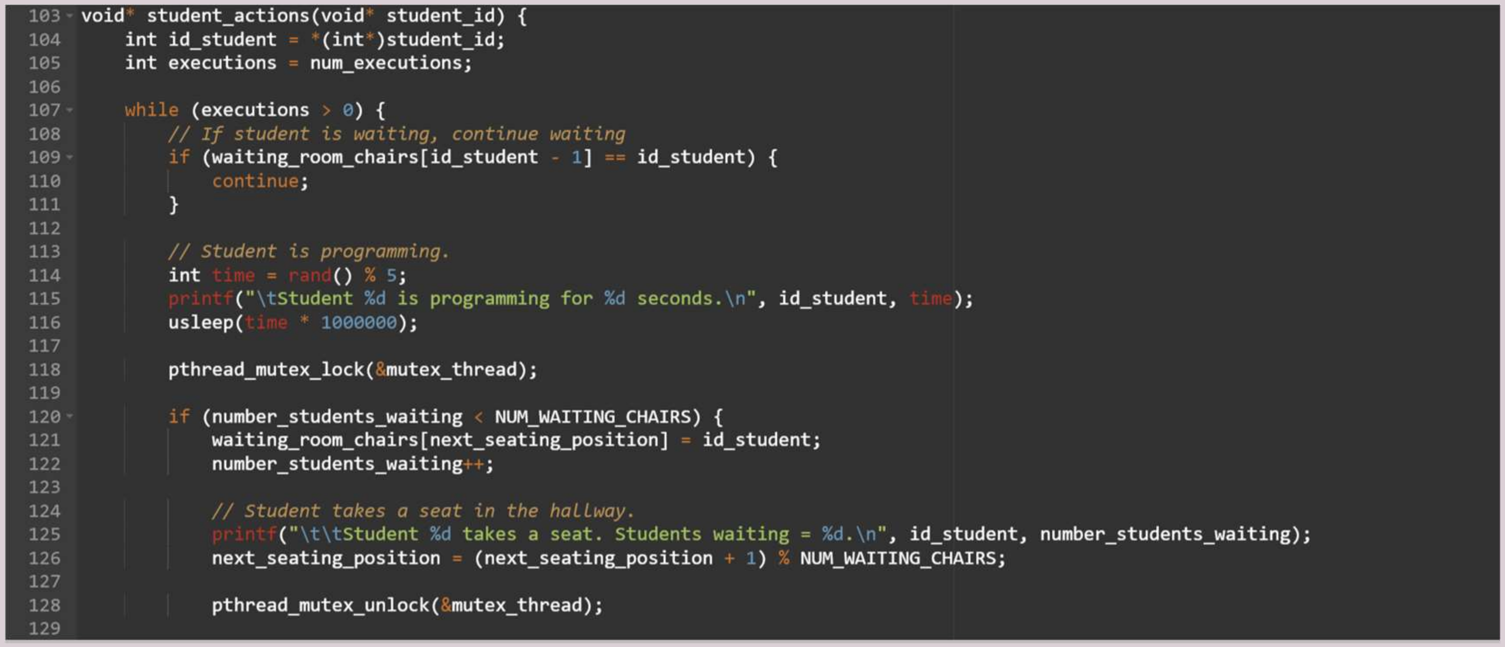
```c
void* ta_actions() {
    printf("Checking for students.\n");

    while (num_executions > 0) {
        // If students are waiting
        if (number_students_waiting > 0) {
            ta_sleeping_flag = 0;
            sem_wait(&sem_students);
            pthread_mutex_lock(&mutex_thread);

            int help_time = rand() % 5;

            // TA helping student.
            printf("Helping a student for %d seconds. Students waiting = %d.\n", help_time, (number_students_waiting - 1));
            printf("Student %d receiving help.\n", waiting_room_chairs[next_teaching_position]);

            waiting_room_chairs[next_teaching_position] = 0;
            number_students_waiting--;
            next_teaching_position = (next_teaching_position + 1) % NUM_WAITING_CHAIRS;

            usleep(help_time * 1000000);

            pthread_mutex_unlock(&mutex_thread);
            sem_post(&sem_ta);

            num_executions--;
        }
```

VOID TEACHING ASSISTANT FUNCTION CODE



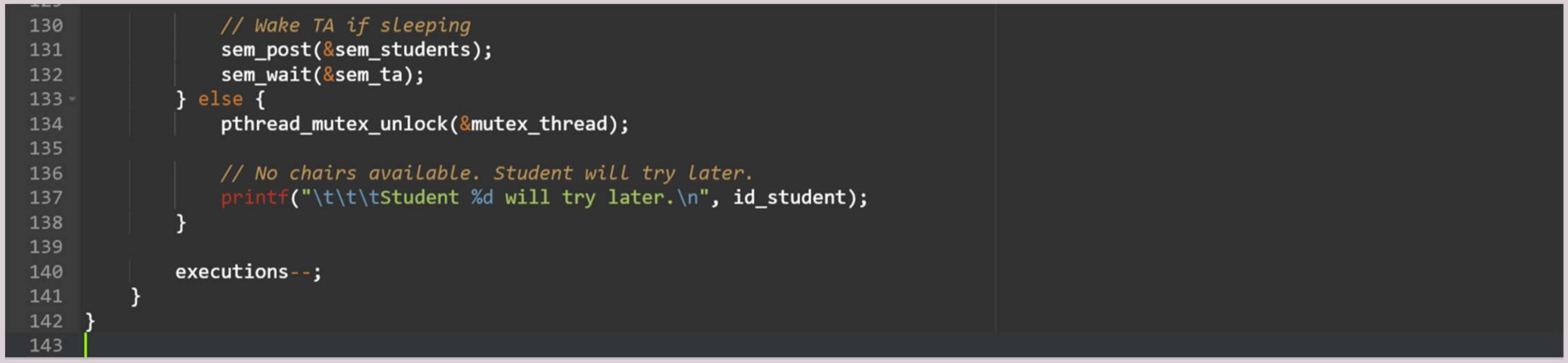
```
        }
        // If no students are waiting
        else {
            if (ta_sleeping_flag == 0) {
                printf("No students waiting. Sleeping.\n");
                ta_sleeping_flag = 1;
            }
        }
    }
}
```

STUDENT FUNCTION CODE



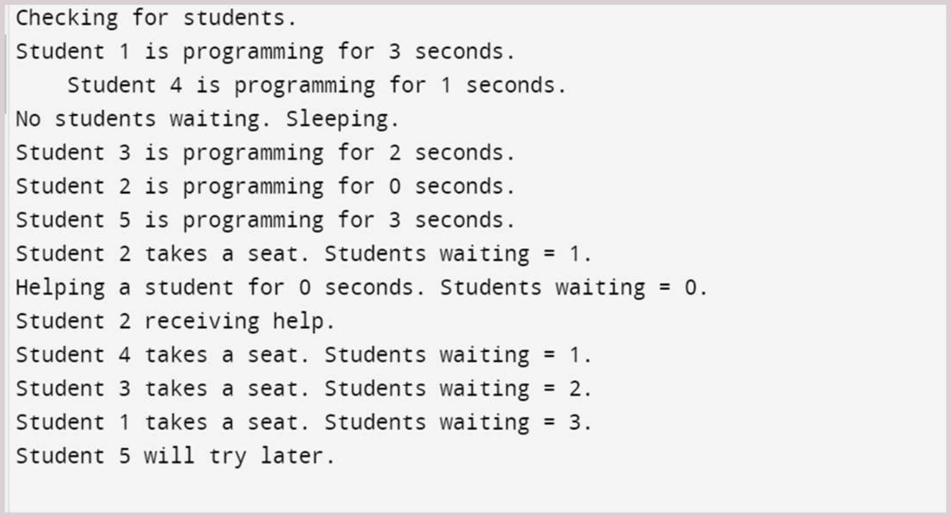
```c
void* student_actions(void* student_id) {
    int id_student = *(int*)student_id;
    int executions = num_executions;

    while (executions > 0) {
        // If student is waiting, continue waiting
        if (waiting_room_chairs[id_student - 1] == id_student) {
            continue;
        }

        // Student is programming.
        int time = rand() % 5;
        printf("\tStudent %d is programming for %d seconds.\n", id_student, time);
        usleep(time * 1000000);

        pthread_mutex_lock(&mutex_thread);

        if (number_students_waiting < NUM_WAITING_CHAIRS) {
            waiting_room_chairs[next_seating_position] = id_student;
            number_students_waiting++;

            // Student takes a seat in the hallway.
            printf("\t\tStudent %d takes a seat. Students waiting = %d.\n", id_student, number_students_waiting);
            next_seating_position = (next_seating_position + 1) % NUM_WAITING_CHAIRS;

            pthread_mutex_unlock(&mutex_thread);

```

STUDENT FUNCTION CODE

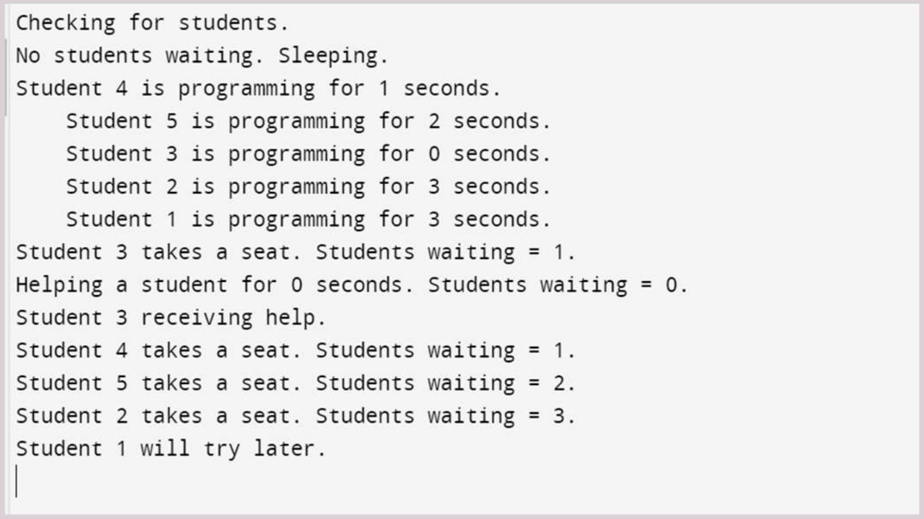


```
            // Wake TA if sleeping
            sem_post(&sem_students);
            sem_wait(&sem_ta);
        } else {
            pthread_mutex_unlock(&mutex_thread);

            // No chairs available. Student will try later.
            printf("\t\t\tStudent %d will try later.\n", id_student);
        }

        executions--;
    }
}
```

OUTPUT



```
Checking for students.
Student 1 is programming for 3 seconds.
    Student 4 is programming for 1 seconds.
No students waiting. Sleeping.
Student 3 is programming for 2 seconds.
Student 2 is programming for 0 seconds.
Student 5 is programming for 3 seconds.
Student 2 takes a seat. Students waiting = 1.
Helping a student for 0 seconds. Students waiting = 0.
Student 2 receiving help.
Student 4 takes a seat. Students waiting = 1.
Student 3 takes a seat. Students waiting = 2.
Student 1 takes a seat. Students waiting = 3.
Student 5 will try later.
```



```
Checking for students.
No students waiting. Sleeping.
Student 4 is programming for 1 seconds.
    Student 5 is programming for 2 seconds.
    Student 3 is programming for 0 seconds.
    Student 2 is programming for 3 seconds.
    Student 1 is programming for 3 seconds.
Student 3 takes a seat. Students waiting = 1.
Helping a student for 0 seconds. Students waiting = 0.
Student 3 receiving help.
Student 4 takes a seat. Students waiting = 1.
Student 5 takes a seat. Students waiting = 2.
Student 2 takes a seat. Students waiting = 3.
Student 1 will try later.
```

SUMMING UP

Thanking you.

Ch Mohith srinivas　　Rushendra Sai　　Jahnavi S