CSE 316

Name- Arindam Banerjee

Section- K21XR   Reg No.- 12112807

Roll No.- K21XRB19

Git hub repository link –

https://github.com/arindambanerjee15/OS_simulation_project.git

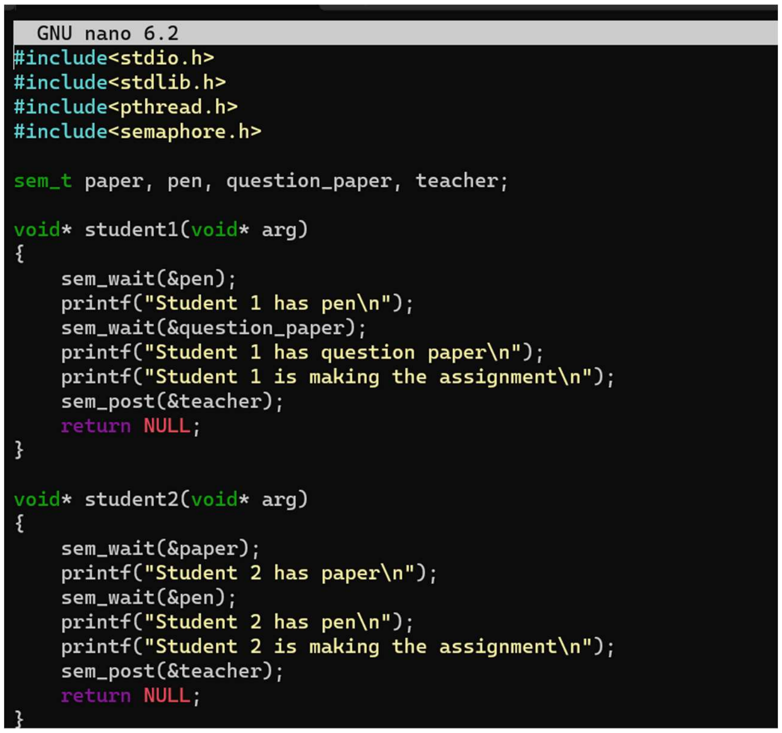
Ques 1- There are 3 student processes and 1 teacher process. Students are supposed to do their assignments and they need 3 things for that-pen, paper and question paper. The teacher has an infinite supply of all the three things. One student has pen, another has paper and another has question paper. The teacher places two things on a shared table and the student having the third complementary thing makes the assignment and tells the teacher on completion. The teacher then places another two things out of the three and again the student having the third thing makes the assignment and tells the teacher on completion. This cycle continues. WAP to synchronise the teacher and the students.

• Two types of people can enter into a library- students and teachers. After entering the library, the visitor searches for the required books and gets them. In order to get them issued, he goes to the single CPU which is there to process the issuing of books. Two types of queues are there at the counter-one for students and one for teachers. A student goes and stands at the tail of the queue for students and similarly the teacher goes and stands at the tail of the queue for teachers (FIFO). If a student is being serviced and a teacher arrives at the counter, he would be the next person to get service (PRIORITY-non preemptive). If two teachers arrive at the same time, they will stand in their queue to get service (FIFO). WAP to ensure that the system works in a non-chaotic manner.

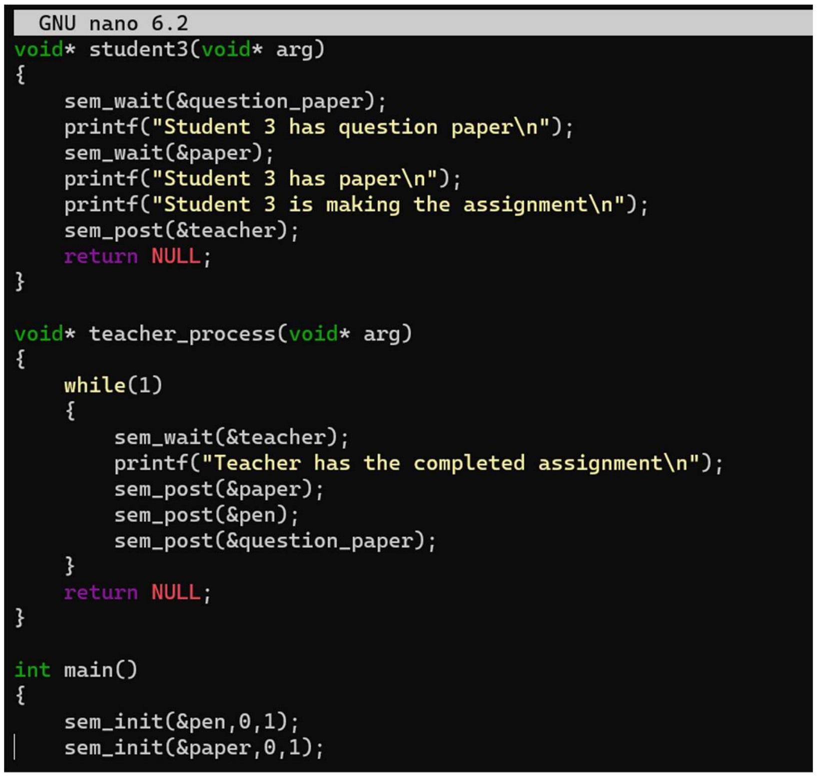
• If a teacher is being served and during the period when he is being served, another teacher comes, then that teacher would get the service next. This process might continue leading to increase in waiting time of students. Ensure in your program that the waiting time of students is minimized.

Solution-

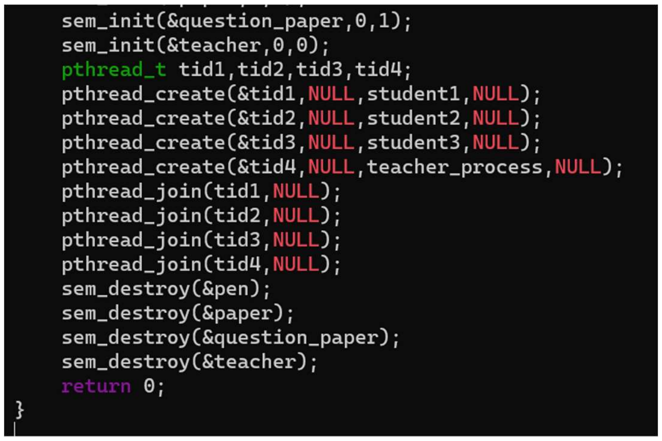
- **Screen shot of the 1st code –**



```
  GNU nano 6.2
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>

sem_t paper, pen, question_paper, teacher;

void* student1(void* arg)
{
    sem_wait(&pen);
    printf("Student 1 has pen\n");
    sem_wait(&question_paper);
    printf("Student 1 has question paper\n");
    printf("Student 1 is making the assignment\n");
    sem_post(&teacher);
    return NULL;
}

void* student2(void* arg)
{
    sem_wait(&paper);
    printf("Student 2 has paper\n");
    sem_wait(&pen);
    printf("Student 2 has pen\n");
    printf("Student 2 is making the assignment\n");
    sem_post(&teacher);
    return NULL;
}
```

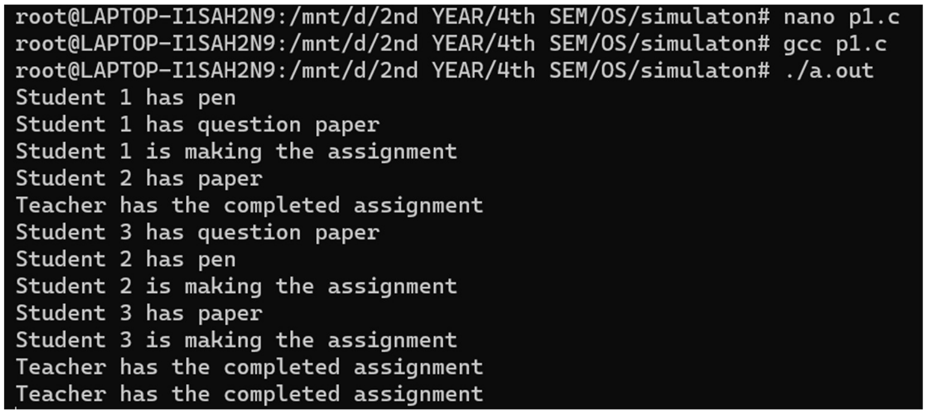


```
  GNU nano 6.2
void* student3(void* arg)
{
    sem_wait(&question_paper);
    printf("Student 3 has question paper\n");
    sem_wait(&paper);
    printf("Student 3 has paper\n");
    printf("Student 3 is making the assignment\n");
    sem_post(&teacher);
    return NULL;
}

void* teacher_process(void* arg)
{
    while(1)
    {
        sem_wait(&teacher);
        printf("Teacher has the completed assignment\n");
        sem_post(&paper);
        sem_post(&pen);
        sem_post(&question_paper);
    }
    return NULL;
}

int main()
{
    sem_init(&pen,0,1);
    sem_init(&paper,0,1);
```



```
    sem_init(&question_paper,0,1);
    sem_init(&teacher,0,0);
    pthread_t tid1,tid2,tid3,tid4;
    pthread_create(&tid1,NULL,student1,NULL);
    pthread_create(&tid2,NULL,student2,NULL);
    pthread_create(&tid3,NULL,student3,NULL);
    pthread_create(&tid4,NULL,teacher_process,NULL);
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    pthread_join(tid3,NULL);
    pthread_join(tid4,NULL);
    sem_destroy(&pen);
    sem_destroy(&paper);
    sem_destroy(&question_paper);
    sem_destroy(&teacher);
    return 0;
}
```

❖ **Screenshot of the output-**

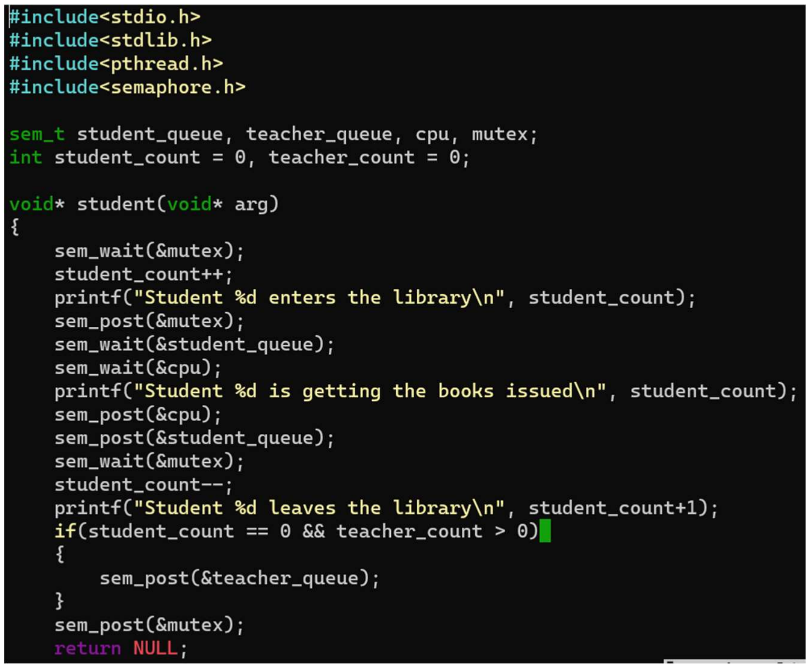


```
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# nano p1.c
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# gcc p1.c
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# ./a.out
Student 1 has pen
Student 1 has question paper
Student 1 is making the assignment
Student 2 has paper
Teacher has the completed assignment
Student 3 has question paper
Student 2 has pen
Student 2 is making the assignment
Student 3 has paper
Student 3 is making the assignment
Teacher has the completed assignment
Teacher has the completed assignment
```

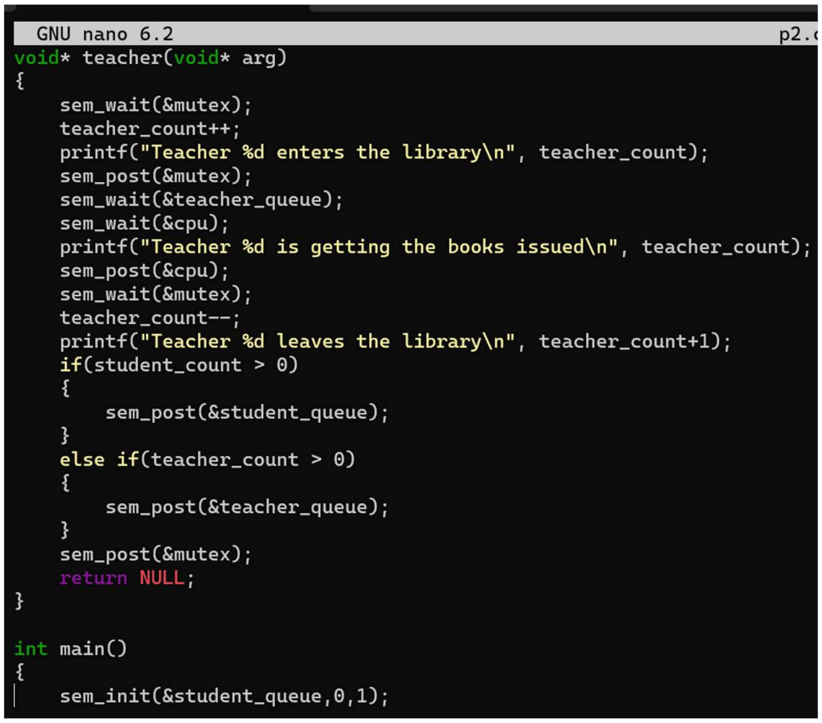
❖ **Explanation of first code:-**

The program uses semaphores to synchronise the teacher and the students. Semaphores are initialised for the three items - pen, paper and question_paper. The teacher semaphore is initialised to 0 since the teacher does not start the process. Then four threads are created - one each for the three students and one for the teacher. The threads for the three students wait on the corresponding semaphores for the two items they need, make the assignment and signal the teacher semaphore. The teacher thread waits on the teacher semaphore, gets the completed assignment and then signals the semaphores for the three items, making them available for the next cycle.

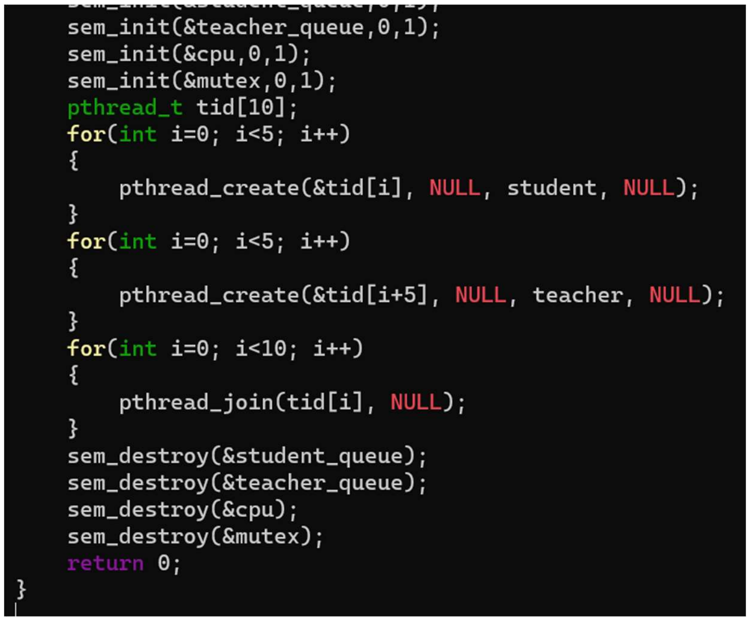
❖ Solution to ensure non-chaotic manner at the library:



```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>

sem_t student_queue, teacher_queue, cpu, mutex;
int student_count = 0, teacher_count = 0;

void* student(void* arg)
{
    sem_wait(&mutex);
    student_count++;
    printf("Student %d enters the library\n", student_count);
    sem_post(&mutex);
    sem_wait(&student_queue);
    sem_wait(&cpu);
    printf("Student %d is getting the books issued\n", student_count);
    sem_post(&cpu);
    sem_post(&student_queue);
    sem_wait(&mutex);
    student_count--;
    printf("Student %d leaves the library\n", student_count+1);
    if(student_count == 0 && teacher_count > 0)
    {
        sem_post(&teacher_queue);
    }
    sem_post(&mutex);
    return NULL;
```

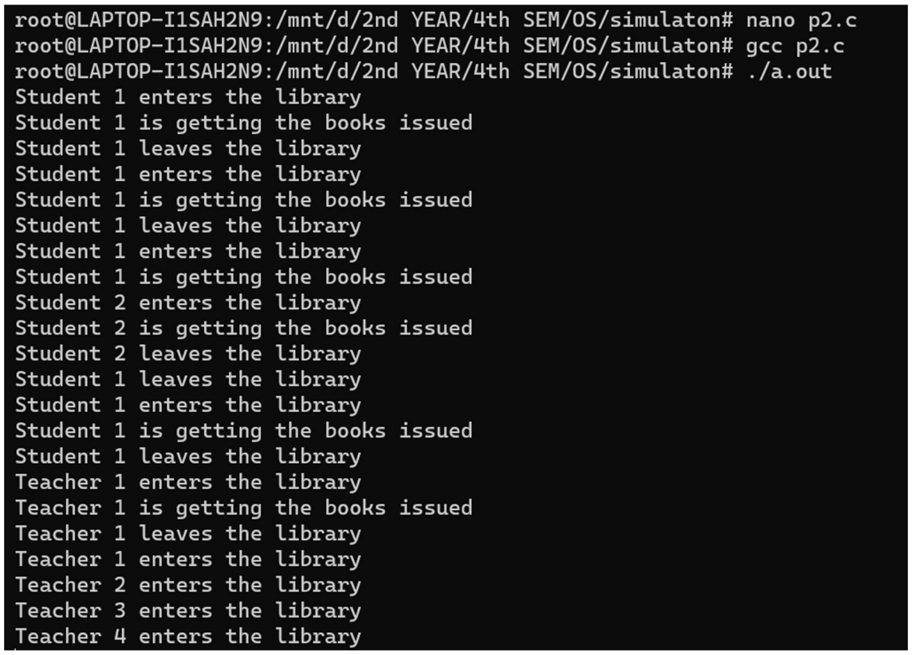


```
  GNU nano 6.2                                                        p2.c
void* teacher(void* arg)
{
    sem_wait(&mutex);
    teacher_count++;
    printf("Teacher %d enters the library\n", teacher_count);
    sem_post(&mutex);
    sem_wait(&teacher_queue);
    sem_wait(&cpu);
    printf("Teacher %d is getting the books issued\n", teacher_count);
    sem_post(&cpu);
    sem_wait(&mutex);
    teacher_count--;
    printf("Teacher %d leaves the library\n", teacher_count+1);
    if(student_count > 0)
    {
        sem_post(&student_queue);
    }
    else if(teacher_count > 0)
    {
        sem_post(&teacher_queue);
    }
    sem_post(&mutex);
    return NULL;
}

int main()
{
    sem_init(&student_queue,0,1);
```



```c
    sem_init(&teacher_queue,0,1);
    sem_init(&cpu,0,1);
    sem_init(&mutex,0,1);
    pthread_t tid[10];
    for(int i=0; i<5; i++)
    {
        pthread_create(&tid[i], NULL, student, NULL);
    }
    for(int i=0; i<5; i++)
    {
        pthread_create(&tid[i+5], NULL, teacher, NULL);
    }
    for(int i=0; i<10; i++)
    {
        pthread_join(tid[i], NULL);
    }
    sem_destroy(&student_queue);
    sem_destroy(&teacher_queue);
    sem_destroy(&cpu);
    sem_destroy(&mutex);
    return 0;
}
```

❖ **Screenshot of output: -**



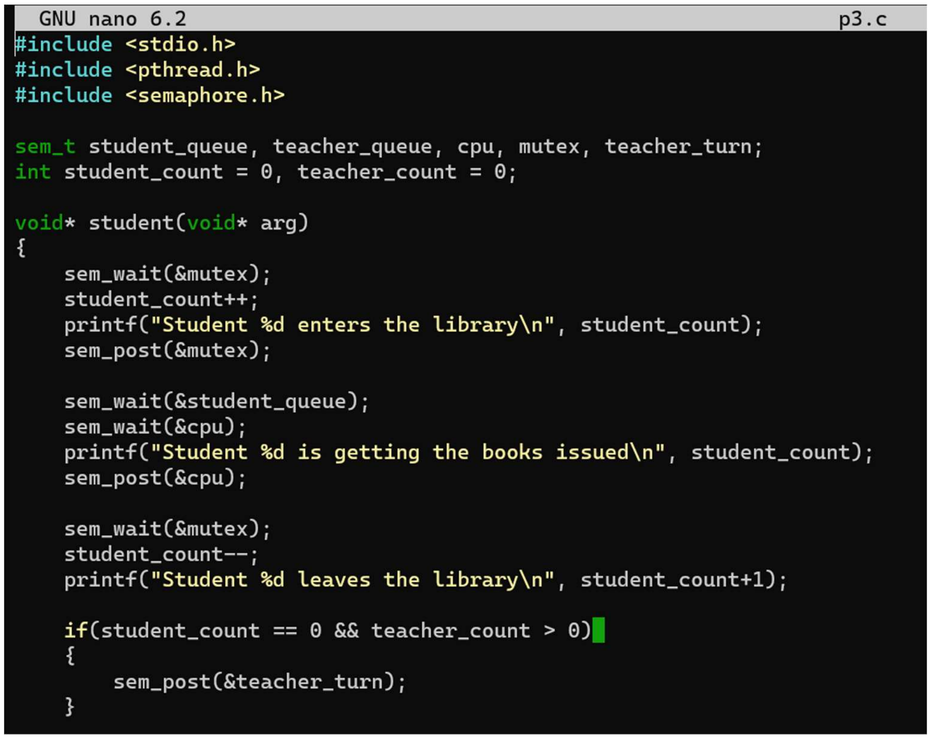
```
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# nano p2.c
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# gcc p2.c
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# ./a.out
Student 1 enters the library
Student 1 is getting the books issued
Student 1 leaves the library
Student 1 enters the library
Student 1 is getting the books issued
Student 1 leaves the library
Student 1 enters the library
Student 1 is getting the books issued
Student 2 enters the library
Student 2 is getting the books issued
Student 2 leaves the library
Student 1 leaves the library
Student 1 enters the library
Student 1 is getting the books issued
Student 1 leaves the library
Teacher 1 enters the library
Teacher 1 is getting the books issued
Teacher 1 leaves the library
Teacher 1 enters the library
Teacher 2 enters the library
Teacher 3 enters the library
Teacher 4 enters the library
```

❖ **Explanation:**

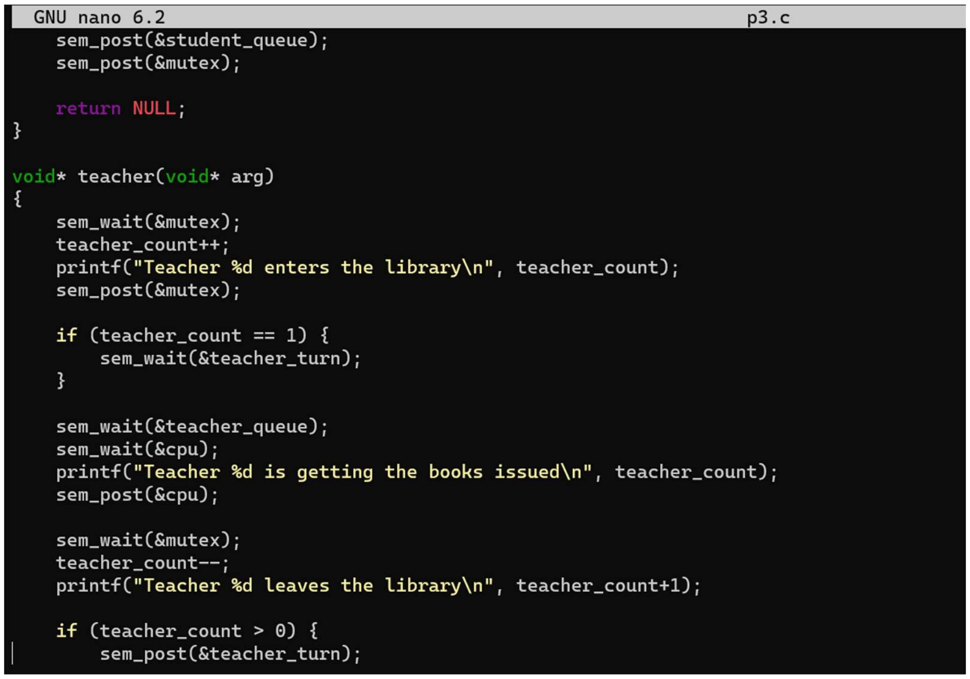
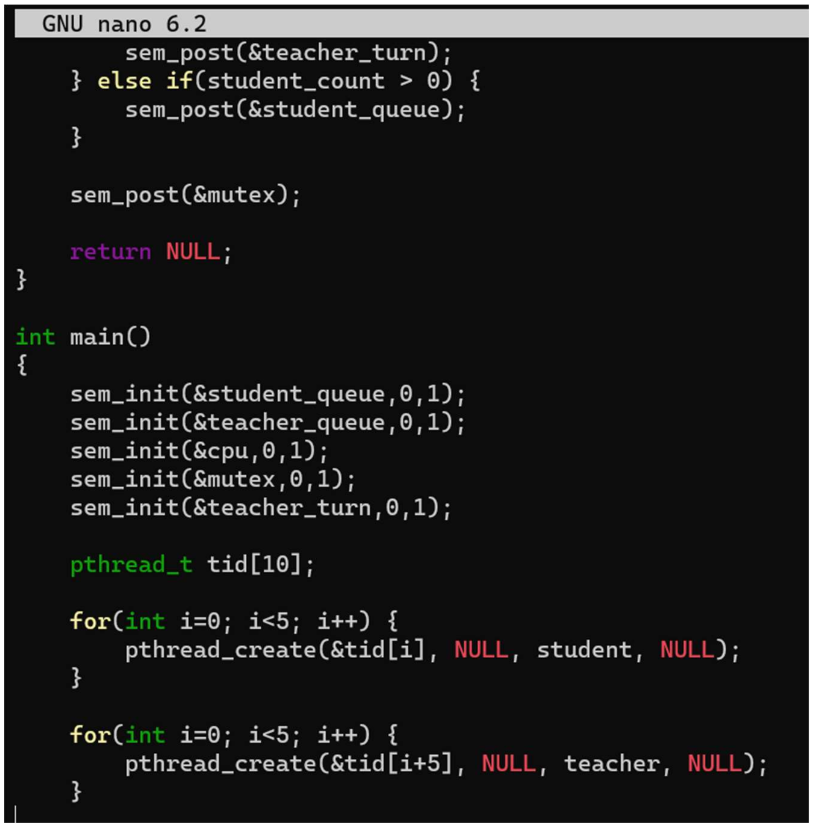
The program uses semaphores to ensure a non-chaotic manner in the library. Semaphores are initialised for the two queues (student and teacher), CPU and a mutex to prevent race conditions. The program creates 5 threads for students and 5 threads for teachers. Each thread first increments the student_count or teacher_count, prints a message and then waits on the corresponding queue semaphore. When a thread gets access to the CPU, it prints a message, decrements the student_count or teacher_count, and checks if there are other students or teachers waiting in the queue. If there are, the appropriate queue semaphore is signaled, else the thread leaves. The mutex is used to prevent race conditions when accessing the shared variables. Finally, the program destroys the semaphores.

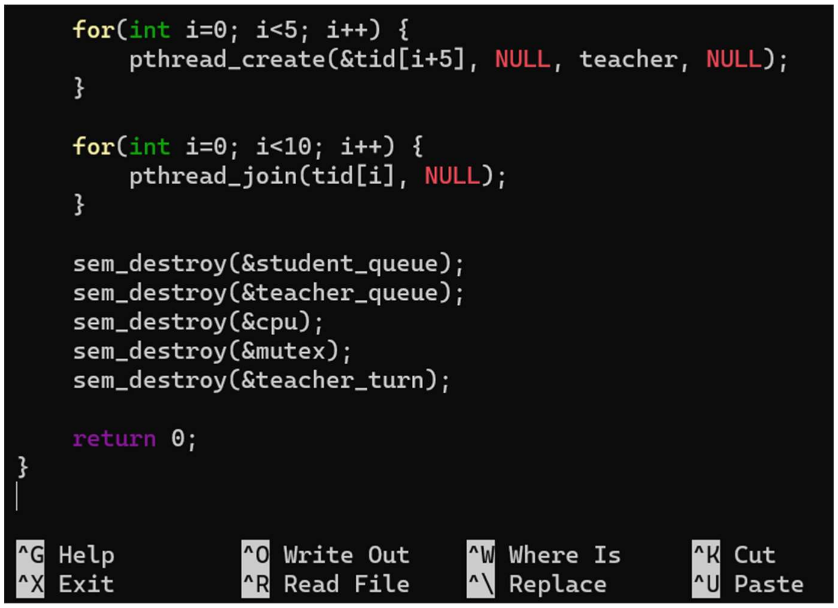
B) To minimize the waiting time of students and ensure that a teacher who arrives while a teacher is being served is served next, we can modify the existing program by introducing an additional semaphore called teacher_turn which ensures that a teacher who arrives while a teacher is being served is given priority.

Screenshot of the code-



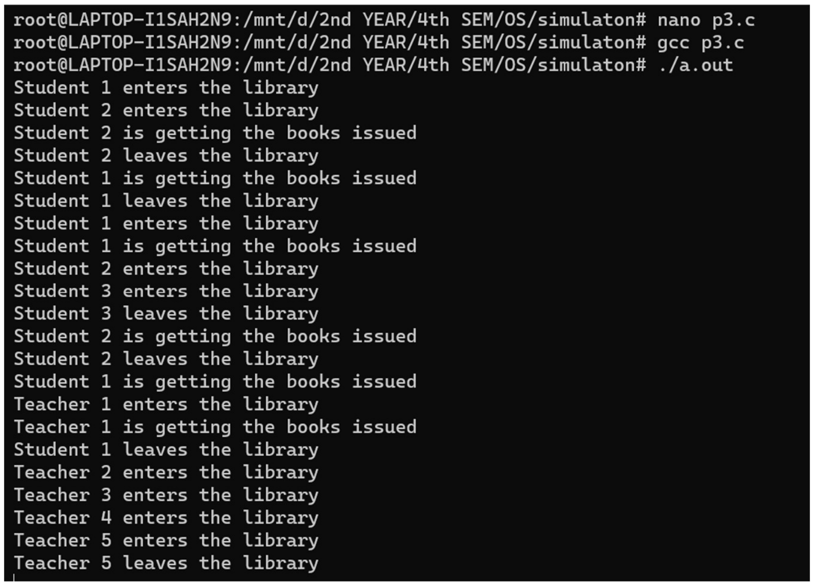
```
  GNU nano 6.2                                                       p3.c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t student_queue, teacher_queue, cpu, mutex, teacher_turn;
int student_count = 0, teacher_count = 0;

void* student(void* arg)
{
    sem_wait(&mutex);
    student_count++;
    printf("Student %d enters the library\n", student_count);
    sem_post(&mutex);

    sem_wait(&student_queue);
    sem_wait(&cpu);
    printf("Student %d is getting the books issued\n", student_count);
    sem_post(&cpu);

    sem_wait(&mutex);
    student_count--;
    printf("Student %d leaves the library\n", student_count+1);

    if(student_count == 0 && teacher_count > 0)
    {
        sem_post(&teacher_turn);
    }
```


```
  GNU nano 6.2                                                   p3.c
    sem_post(&student_queue);
    sem_post(&mutex);

    return NULL;
}

void* teacher(void* arg)
{
    sem_wait(&mutex);
    teacher_count++;
    printf("Teacher %d enters the library\n", teacher_count);
    sem_post(&mutex);

    if (teacher_count == 1) {
        sem_wait(&teacher_turn);
    }

    sem_wait(&teacher_queue);
    sem_wait(&cpu);
    printf("Teacher %d is getting the books issued\n", teacher_count);
    sem_post(&cpu);

    sem_wait(&mutex);
    teacher_count--;
    printf("Teacher %d leaves the library\n", teacher_count+1);

    if (teacher_count > 0) {
        sem_post(&teacher_turn);
```

```
  GNU nano 6.2
        sem_post(&teacher_turn);
    } else if(student_count > 0) {
        sem_post(&student_queue);
    }

    sem_post(&mutex);

    return NULL;
}

int main()
{
    sem_init(&student_queue,0,1);
    sem_init(&teacher_queue,0,1);
    sem_init(&cpu,0,1);
    sem_init(&mutex,0,1);
    sem_init(&teacher_turn,0,1);

    pthread_t tid[10];

    for(int i=0; i<5; i++) {
        pthread_create(&tid[i], NULL, student, NULL);
    }

    for(int i=0; i<5; i++) {
        pthread_create(&tid[i+5], NULL, teacher, NULL);
    }
```



```
    for(int i=0; i<5; i++) {
        pthread_create(&tid[i+5], NULL, teacher, NULL);
    }

    for(int i=0; i<10; i++) {
        pthread_join(tid[i], NULL);
    }

    sem_destroy(&student_queue);
    sem_destroy(&teacher_queue);
    sem_destroy(&cpu);
    sem_destroy(&mutex);
    sem_destroy(&teacher_turn);

    return 0;
}

^G Help          ^O Write Out     ^W Where Is      ^K Cut
^X Exit          ^R Read File     ^\ Replace       ^U Paste
```

Screenshot of the output-



```
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# nano p3.c
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# gcc p3.c
root@LAPTOP-I1SAH2N9:/mnt/d/2nd YEAR/4th SEM/OS/simulaton# ./a.out
Student 1 enters the library
Student 2 enters the library
Student 2 is getting the books issued
Student 2 leaves the library
Student 1 is getting the books issued
Student 1 leaves the library
Student 1 enters the library
Student 1 is getting the books issued
Student 2 enters the library
Student 3 enters the library
Student 3 leaves the library
Student 2 is getting the books issued
Student 2 leaves the library
Student 1 is getting the books issued
Teacher 1 enters the library
Teacher 1 is getting the books issued
Student 1 leaves the library
Teacher 2 enters the library
Teacher 3 enters the library
Teacher 4 enters the library
Teacher 5 enters the library
Teacher 5 leaves the library
```

Explanation:

In this modified program, we introduce an additional semaphore teacher_turn. When a teacher arrives and finds another teacher already being served, the arriving teacher waits on the teacher_turn semaphore instead of the teacher_queue semaphore. This ensures that the arriving teacher will be served next after the currently being served teacher is done.

The rest of the program remains the same as the original program.