EAST WEST UNIVERSITY

Course Title	**:**	Operating Systems
Course Code	**:**	CSE325
Section	**:**	(04)
Semester	**:**	Fall 2021

Project Report

Project Name: Seeking Tutor Problem

Submitted To

Masiath Mubassira
Lecturer
Department of Computer Science and Engineering
East West University

Submitted By

Md. Jannatul Haq	2018-1-60-224
Mehzabin Meem	2019-2-60-019
Md. Wahiduzzaman	2019-2-60-048
Labonno Khan Bonna	2019-2-60-044

Submission Date

21/01/2022

Contents:

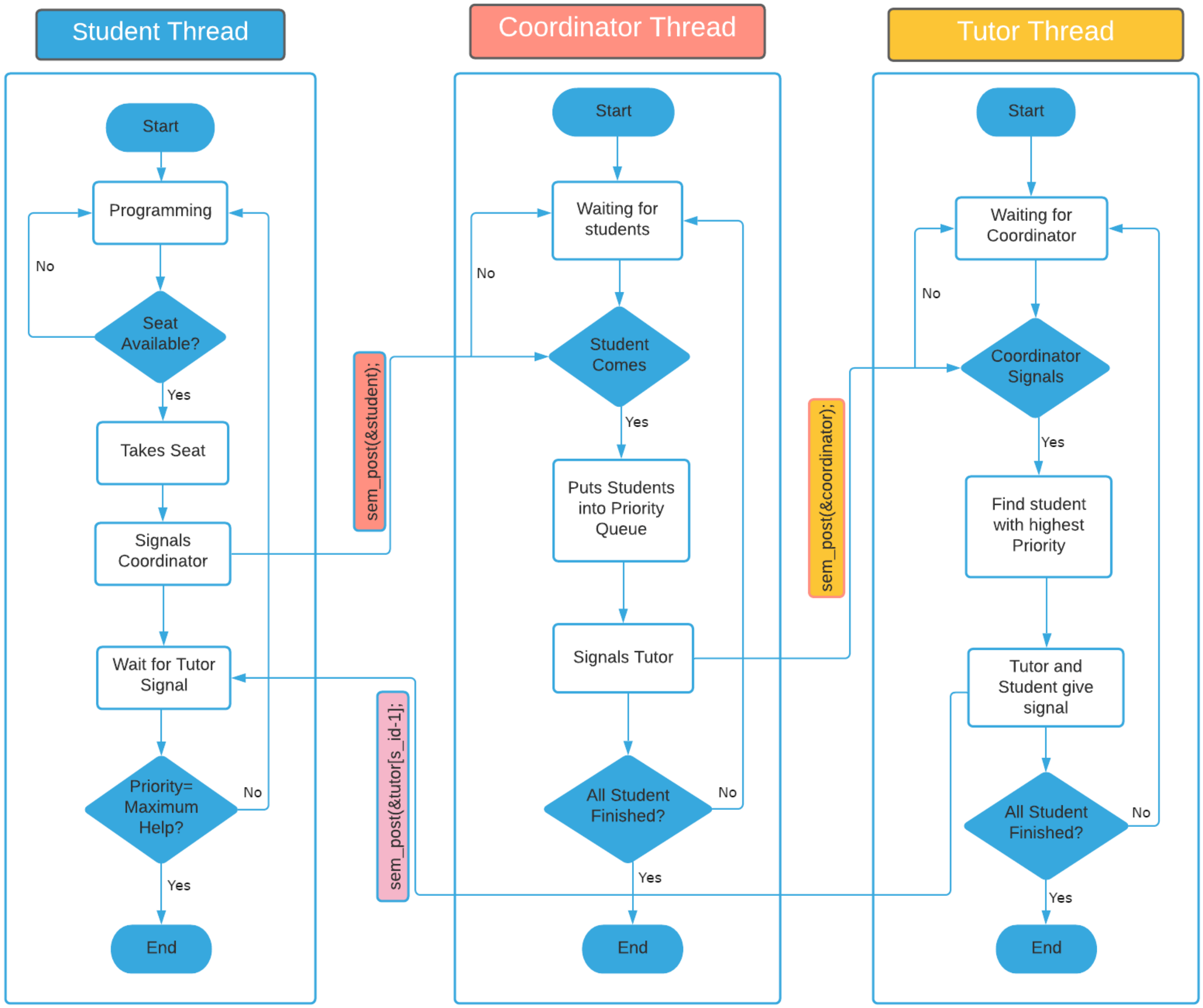
Project Description:

We have a club where undergraduate students get help with their programming assignments from tutors. The club has a coordinator and several tutors to assist the students. Here, students can come to the waiting area, if there is no available seat then they will go back to programming. Otherwise, they will wait in the waiting area. The coordinator will then prioritize students in a queue. Then tutor will find the student from the queue who has the highest priority and tutor them. If more than one student has the same priority the tutor will choose who came first. After getting the maximum amount of help students terminate. When all students will be finished getting help, the tutor will terminate and then the coordinator will be terminated.

Methodologies we have used in our code:

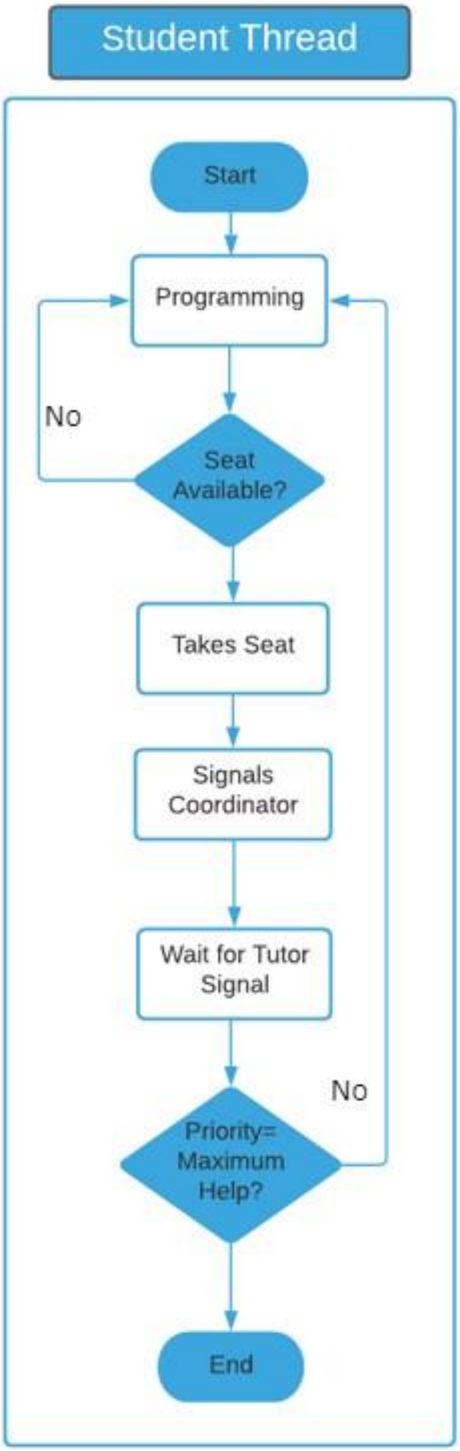
- Pthread_create()
- Pthread_join()
- Sem_init()
- Sem_wait()
- Sem_post()
- Sem_mutex()

Flowchart:


Student Thread
Start
Programming
No
Seat Available?
Yes
Takes Seat
Signals Coordinator
Wait for Tutor Signal
Priority= Maximum Help?
No
Yes
End
sem_post(&student);
sem_post(&tutor[s_id-1]);
Coordinator Thread
Start
Waiting for students
No
Student Comes
Yes
Puts Students into Priority Queue
Signals Tutor
All Student Finished?
No
Yes
End
sem_post(&coordinator);
Tutor Thread
Start
Waiting for Coordinator
No
Coordinator Signals
Yes
Find student with highest Priority
Tutor and Student give signal
All Student Finished?
No
Yes
End

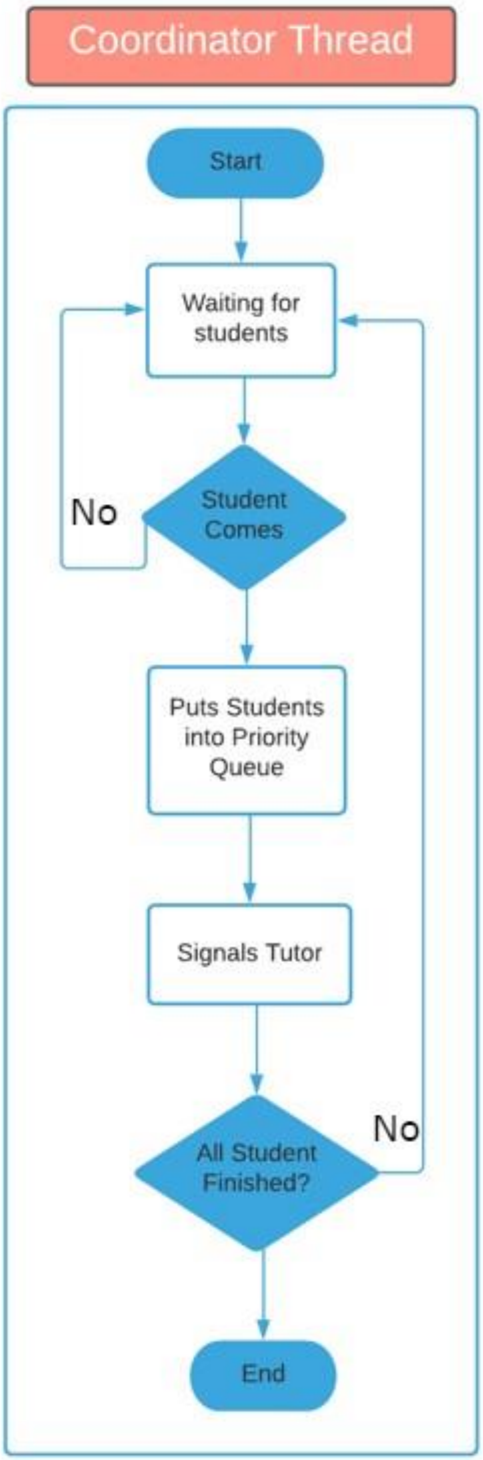
Description of each Function:

The student has three parts:



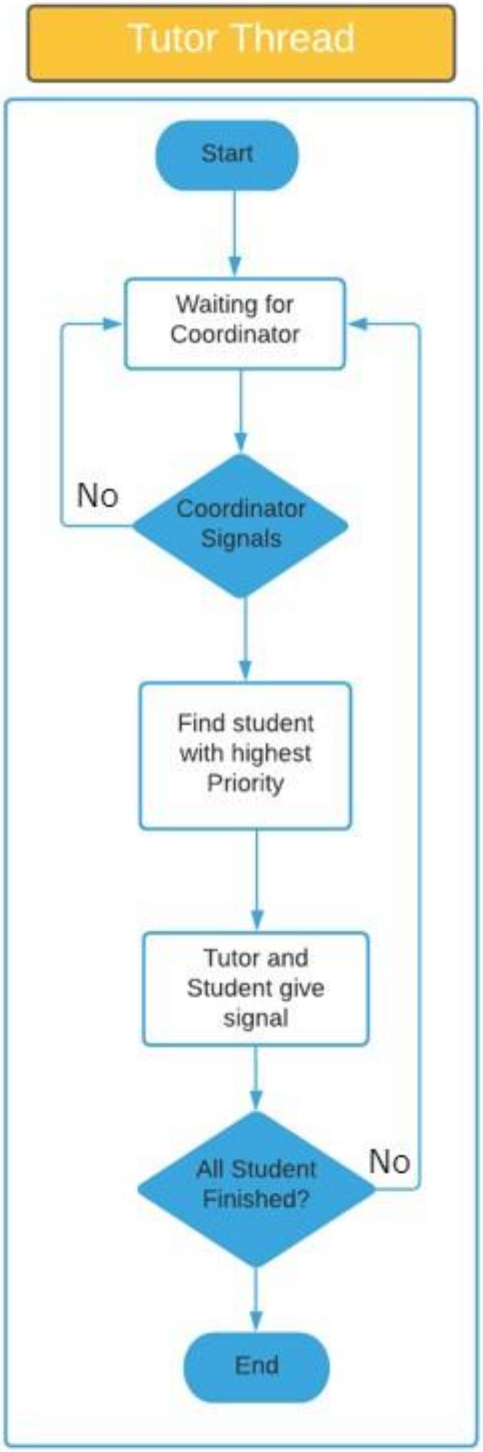
1. Student has to start programming and seek help from a tutor, the student needs to get a seat. After coming to get seat, if student does not find any seat, then has to go back to programming and try again later.
2. If a student gets a seat, then the coordinator gets a notice and then waits for a tutor to be assigned.
3. After getting the maximum amount of help, the student will stop seeking help.

► **Coordinate has 4 parts:**



1. The coordinator waits for students to come to seek help.
2. And prioritize students according to who has come first.
3. Then coordinator assigns tutors to each student.
4. If all students in the waiting area get a tutor, then the coordinator notifies the tutor and leaves.

Tutor has 3 parts:



1. Tutor waits for the coordinator to notify if a student has come to seek help.
2. Tutor selects to the student to help according to the priority of the student.
3. If all students have done seeking help from the tutor, the tutor waits for coordinators notification to finish work.

Code:

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

struct priority_queue
{
int priority_no;
int time;
};

int finished_student_no=0, finished_tutor_no = 0, request_no = 0, student_no=0,
tutor_no=0, help = 0, total_chair = 0, occupied_chairs=0;

int visited[100];
struct priority_queue pq[100];
int priority[100];
int student_ids[100];
int tutor_ids[100];

sem_t student;
sem_t coordinator;
sem_t tutor[100];
sem_t mutex;

void *student_thread(void *student_id)
```

```
{
   int s_id=*(int*)student_id;

   while(1)
        {
      if(priority[s_id-1] == help)
                {

                        sem_wait(&mutex);
         finished_student_no++;
                        sem_post(&mutex);

         printf("\n\nstudent %d terminates\n\n",s_id);

         sem_post(&student);
         pthread_exit(NULL);
      }



                sem_wait(&mutex);
      if(occupied_chairs == total_chair)
                {
         printf("\nStudent: Student %d found no empty chair.\n",s_id);
                        sem_post(&mutex);
         continue;
      }
```

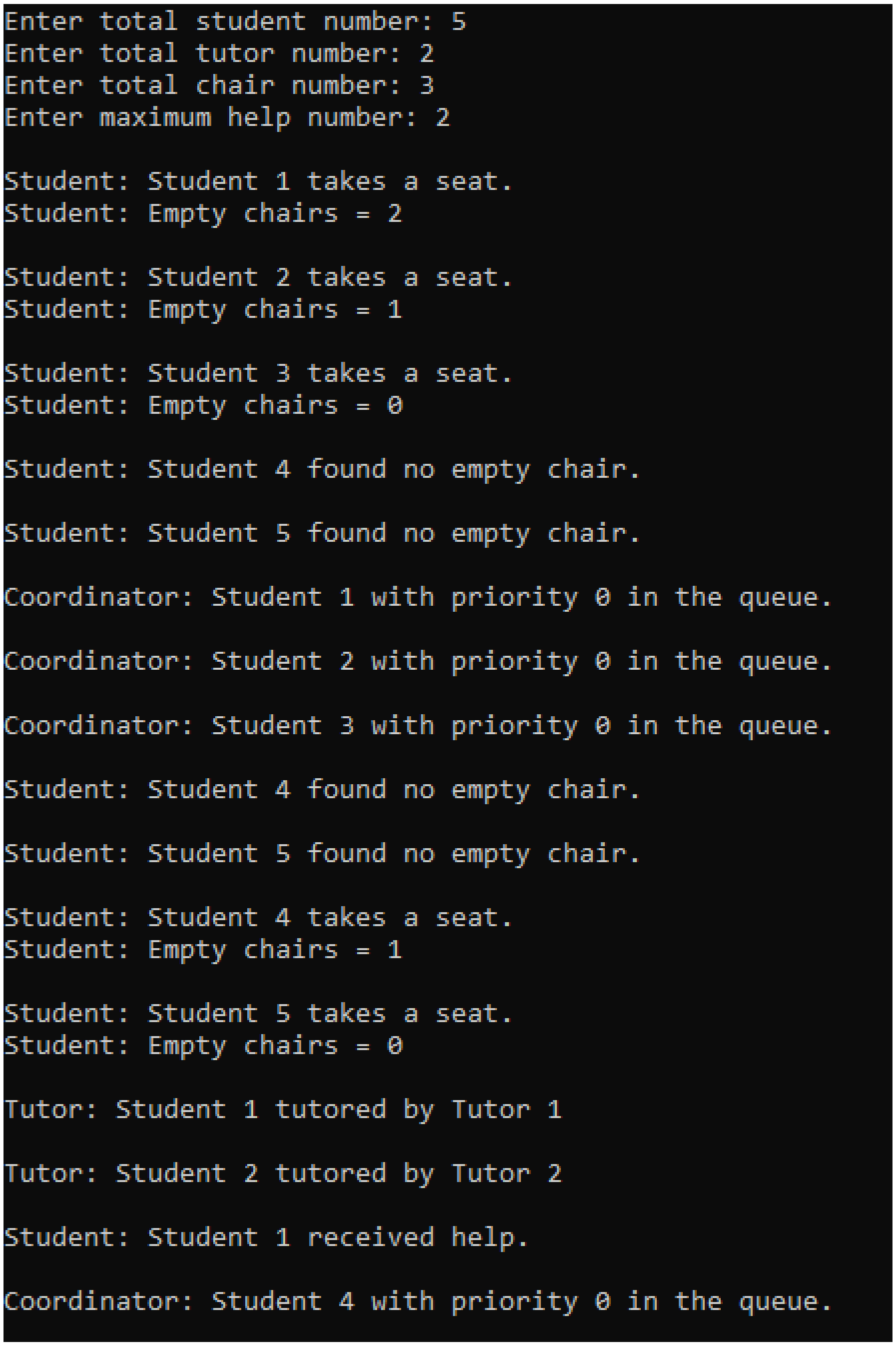
```
        occupied_chairs++;
        request_no++;
        visited[s_id-1]=request_no;
        printf("\nStudent: Student %d takes a seat.\nStudent: Empty chairs = 
%d\n",s_id,total_chair-occupied_chairs);
                sem_post(&mutex);


        sem_post(&student);
                sem_wait(&tutor[s_id-1]);

        printf("\nStudent: Student %d received help.\n",s_id);

                sem_wait(&mutex);
        priority[s_id-1]++;
                printf("\nStudent: Student %d priority now is %d\n",s_id, 
priority[s_id-1]);
                sem_post(&mutex);
    }
}


void *tutor_thread(void *tutor_id)
{
      int t_id=*(int*)tutor_id;

    while(1)
```

```
   {
 if(finished_student_no==student_no)
            {
 sem_wait(&mutex);
 finished_tutor_no++;
 sem_post(&mutex);

 sem_wait(&mutex);
     printf("\n\ntutor %d terminates\n\n",t_id);
     if(finished_tutor_no == tutor_no)
     {
     printf("\n\ncoordinator terminates\n\n");


     }
     sem_post(&mutex);

     pthread_exit(NULL);
 }



 sem_wait(&coordinator);


int max_request=student_no*help+1, max_priority = help-1 ,s_id = -1;
            sem_wait(&mutex);
for(int i=0;i<student_no;i++)
            {
     if(pq[i].priority_no>-1 && pq[i].priority_no<=max_priority)
```

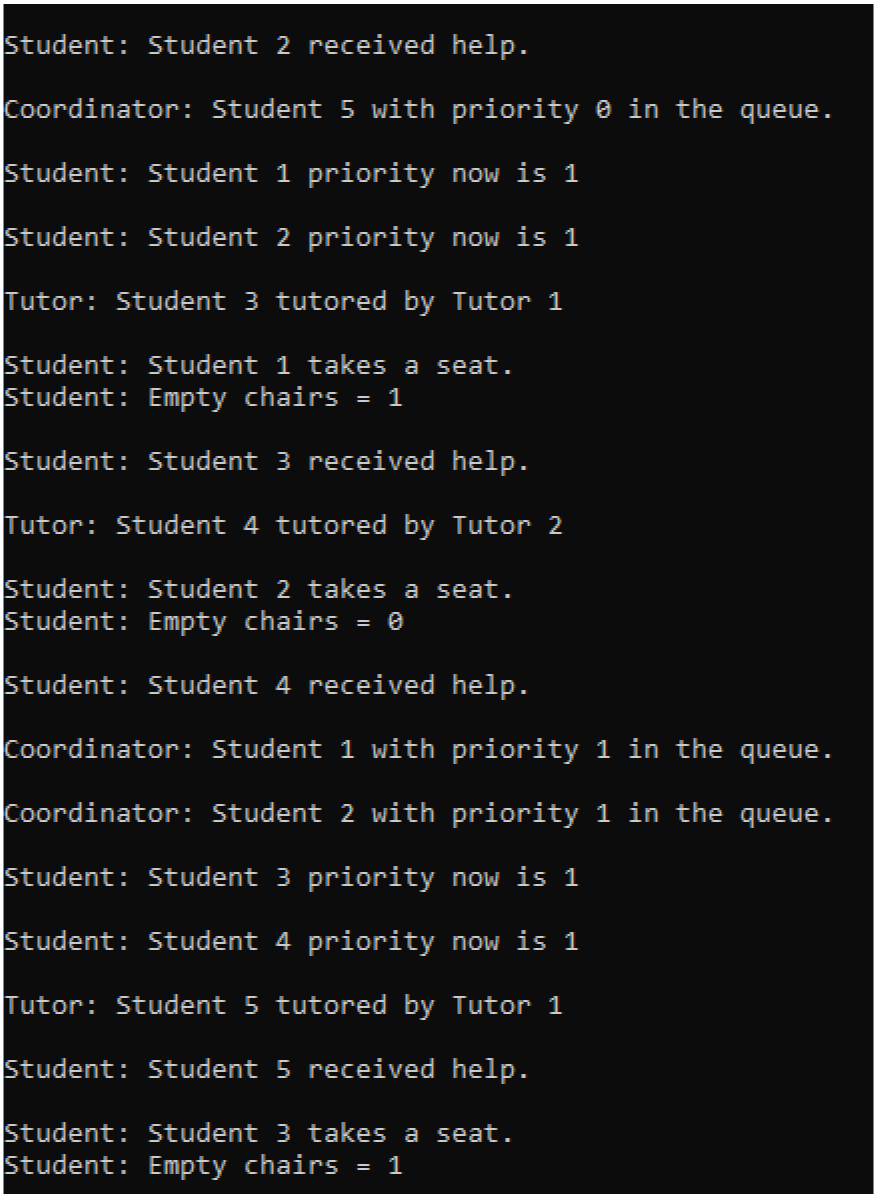
```
               {
        if (pq[i].time<max_request)
             {
               max_priority = pq[i].priority_no;
               max_request=pq[i].time;
               s_id=student_ids[i];
              }
           }
          }

if(s_id==-1)
        {
               sem_post(&mutex);
   continue;
}

        pq[s_id-1].priority_no = -1;
        pq[s_id-1].time = -1;

occupied_chairs--;
        sem_post(&mutex);

        sem_wait(&mutex);
printf("\nTutor: Student %d tutored by Tutor %d\n",s_id,t_id);
        sem_post(&mutex);

        sem_post(&tutor[s_id-1]);
```

```
    }
}


void *coordinator_thread()
{
    while(1)
          {
        if(finished_student_no==student_no)
                     {
            for(int i=0;i<tutor_no;i++)
                               {
                  sem_post(&coordinator);
            }
            pthread_exit(NULL);
        }

        sem_wait(&student);

                     sem_wait(&mutex);
        for(int i=0;i<student_no;i++)
                     {
            if(visited[i]>-1)
                               {
                pq[i].priority_no = priority[i];
                pq[i].time = visited[i];
        printf("\nCoordinator: Student %d with priority %d in the
queue.\n",student_ids[i],priority[i]);
        visited[i]=-1;

        sem_post(&coordinator);
      }
    }
          sem_post(&mutex);
  }
}



int main()
{
     printf("Enter total student number: ");
     scanf("%d", &student_no);
     printf("Enter total tutor number: ");
     scanf("%d", &tutor_no);
     printf("Enter total chair number: ");
     scanf("%d", &total_chair);
     printf("Enter maximum help number: ");
     scanf("%d", &help);

  for(int i=0;i<student_no;i++)
     {
        visited[i]=-1;
        pq[i].priority_no = -1;
        pq[i].time = -1;
        priority[i]=0;
    }

    sem_init(&student,0,0);
    sem_init(&coordinator,0,0);
          sem_init(&mutex,0,1);
          for(int i=0;i<student_no;i++)
          {
        sem_init(&tutor[i],0,0);
          }

    pthread_t students[student_no];
    pthread_t tutors[tutor_no];
    pthread_t coordinator;

    for(int i = 0; i < student_no; i++)
    {
        student_ids[i] = i + 1;
                    if (pthread_create(&students[i], NULL, student_thread, (void*)
&student_ids[i]) < 0)
        {
            perror("Error: thread cannot be created");
           exit(1);
        }
    }

    for(int i = 0; i < tutor_no; i++)
    {
        tutor_ids[i] = i + 1;
                    if (pthread_create(&tutors[i], NULL, tutor_thread, (void*)
&tutor_ids[i]) < 0)
        {
            perror("Error: thread cannot be created");
           exit(1);
        }
    }

          if (pthread_create(&coordinator,NULL,coordinator_thread,NULL) < 0)
        {
            perror("Error: thread cannot be created");
           exit(1);
        }



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

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


          pthread_join(coordinator, NULL);


    return 0;
}
```

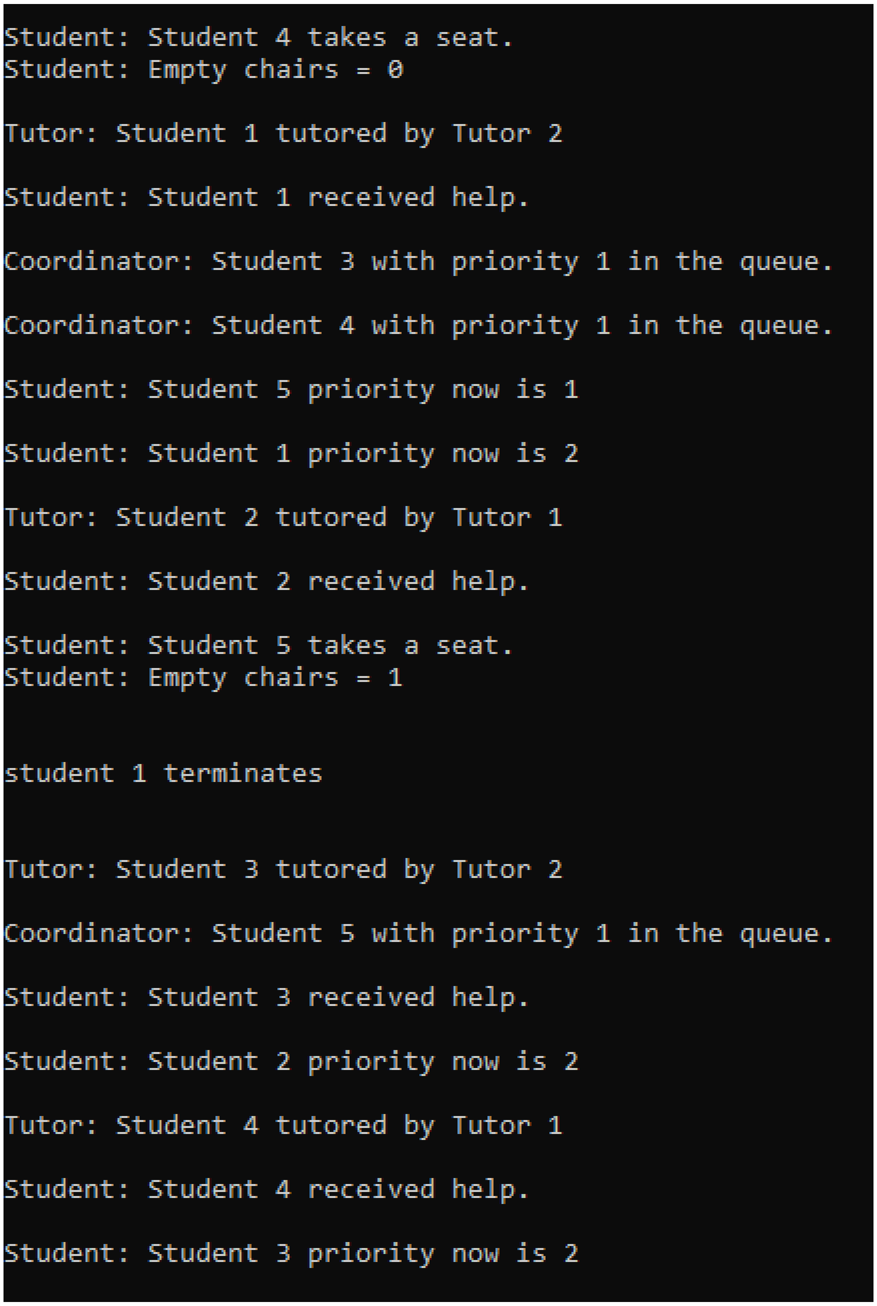
Output:



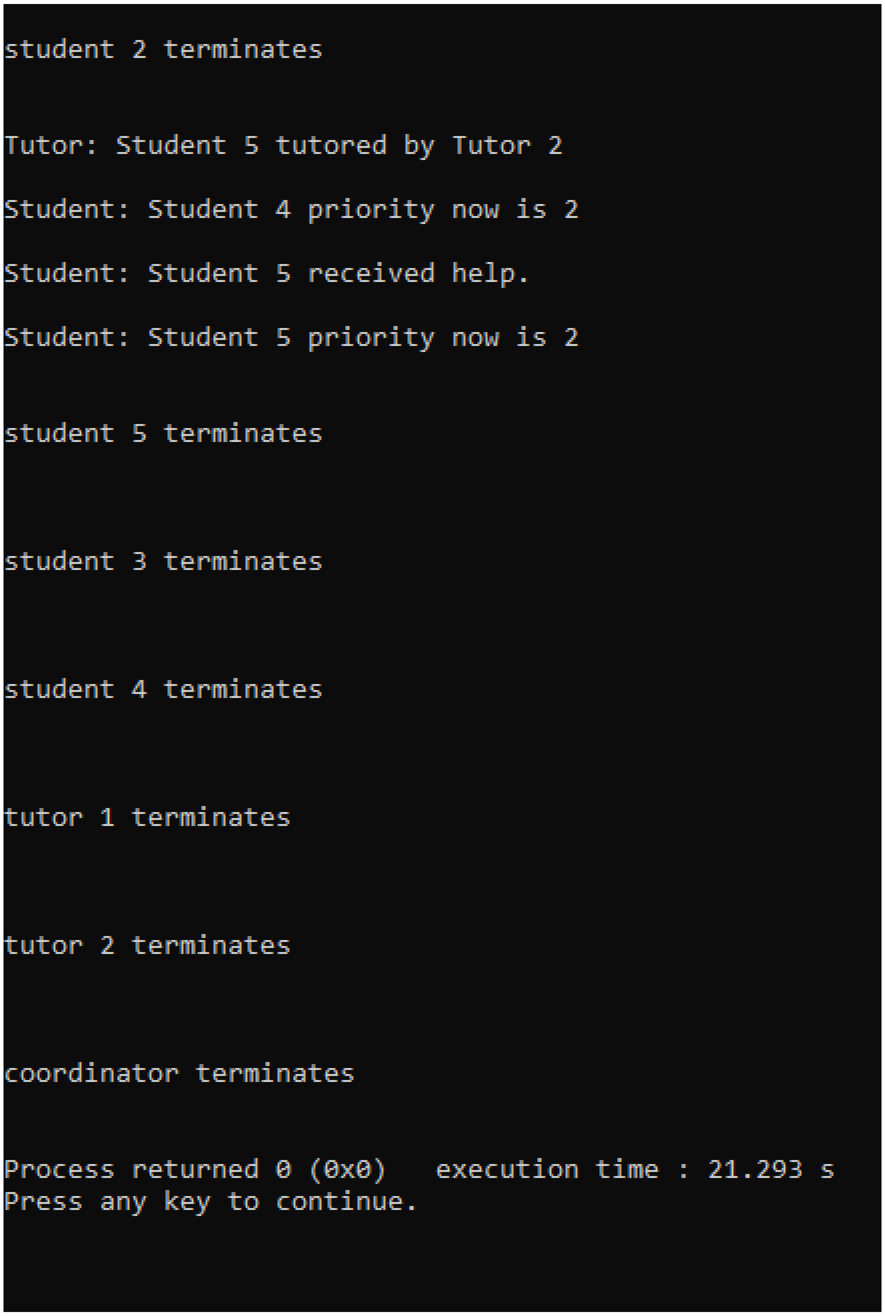
```
Enter total student number: 5
Enter total tutor number: 2
Enter total chair number: 3
Enter maximum help number: 2

Student: Student 1 takes a seat.
Student: Empty chairs = 2

Student: Student 2 takes a seat.
Student: Empty chairs = 1

Student: Student 3 takes a seat.
Student: Empty chairs = 0

Student: Student 4 found no empty chair.

Student: Student 5 found no empty chair.

Coordinator: Student 1 with priority 0 in the queue.

Coordinator: Student 2 with priority 0 in the queue.

Coordinator: Student 3 with priority 0 in the queue.

Student: Student 4 found no empty chair.

Student: Student 5 found no empty chair.

Student: Student 4 takes a seat.
Student: Empty chairs = 1

Student: Student 5 takes a seat.
Student: Empty chairs = 0

Tutor: Student 1 tutored by Tutor 1

Tutor: Student 2 tutored by Tutor 2

Student: Student 1 received help.

Coordinator: Student 4 with priority 0 in the queue.
```



```
Student: Student 2 received help.

Coordinator: Student 5 with priority 0 in the queue.

Student: Student 1 priority now is 1

Student: Student 2 priority now is 1

Tutor: Student 3 tutored by Tutor 1

Student: Student 1 takes a seat.
Student: Empty chairs = 1

Student: Student 3 received help.

Tutor: Student 4 tutored by Tutor 2

Student: Student 2 takes a seat.
Student: Empty chairs = 0

Student: Student 4 received help.

Coordinator: Student 1 with priority 1 in the queue.

Coordinator: Student 2 with priority 1 in the queue.

Student: Student 3 priority now is 1

Student: Student 4 priority now is 1

Tutor: Student 5 tutored by Tutor 1

Student: Student 5 received help.

Student: Student 3 takes a seat.
Student: Empty chairs = 1
```



```
Student: Student 4 takes a seat.
Student: Empty chairs = 0

Tutor: Student 1 tutored by Tutor 2

Student: Student 1 received help.

Coordinator: Student 3 with priority 1 in the queue.

Coordinator: Student 4 with priority 1 in the queue.

Student: Student 5 priority now is 1

Student: Student 1 priority now is 2

Tutor: Student 2 tutored by Tutor 1

Student: Student 2 received help.

Student: Student 5 takes a seat.
Student: Empty chairs = 1


student 1 terminates


Tutor: Student 3 tutored by Tutor 2

Coordinator: Student 5 with priority 1 in the queue.

Student: Student 3 received help.

Student: Student 2 priority now is 2

Tutor: Student 4 tutored by Tutor 1

Student: Student 4 received help.

Student: Student 3 priority now is 2
```



```
student 2 terminates


Tutor: Student 5 tutored by Tutor 2

Student: Student 4 priority now is 2

Student: Student 5 received help.

Student: Student 5 priority now is 2


student 5 terminates



student 3 terminates



student 4 terminates



tutor 1 terminates



tutor 2 terminates



coordinator terminates


Process returned 0 (0x0)   execution time : 21.293 s
Press any key to continue.
```

Conclusion:

This program synchronizes the tasks between the students, coordinator, and tutors. The students get help according to their priorities, so no one is left out.