**East West University**

**Department of CSE**

|  |  |
| --- | --- |
| **CSE-325**  **Operating System** | |
| **Project no:01** | |
| **Project name:**  **Seeking Tutor Problem** | |
| **GROUP NO: 01** | **Course Instructor information :** |
| **Name of Student & Id :**   * **Mohammed Tasiful Alam**   **( 2018-1-60-171)**   * **Tanvir Hasan**   **(2018-1-60-236)** | **Tanni Mitra**  **Lecturer**  **Department**  **of C.S.E**  **East West University** |
| **Date of Submitted :**  **15.01.2021** | |

**PROJECT REPORT**

# CONTENTS

* Problem Statement
* Description of each function
* Code
* Output
* Conclusion

## Problem Statement:

We have a club where undergraduate students get help with their programming assignments from tutor .The club has a coordinator and several tutors to assist the students. Here, students can come to waiting area if there is no available seat 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 whoever came first . After getting the maximum amount of help student terminate. When all students will be finished getting help, tutor will terminate and then coordinator will be terminated.

### Methodologies we have used in the code:

* + Pthread\_create()
  + Pthread\_join()
  + Sem\_init()
  + Sem\_wait()
  + Sem\_post()
  + Sem\_mutex()

**Description of each Function:**

### Student has 3 parts:

1. Student has to start programming and seek help from a tutor ,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.

1. If a student gets a seat then coordinator gets a notice and then wait for tutor to be assigned
2. After getting maximum amount of help, the student will stop seeking help .

### Coordinator 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 tutor to each student.
4. If all students in the waiting area gets a tutor then coordinator notify 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 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;//student's pariority will go

max\_request=pq[i].time;//student's arrival time

s\_id=student\_ids[i];//saving student id in the array

}

}

}

if(s\_id==-1) //found no students

{

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(void \*arg)

{

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];//priority queue time will be the visited time

printf("\nCoordinator: Student %d with priority %d in the queue.\n",student\_ids[i],priority[i]);

visited[i]=-1;

sem\_post(&coordinator);//coordinator passes the signal to tutor

}

}

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);// only for critical secison we have taken 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);

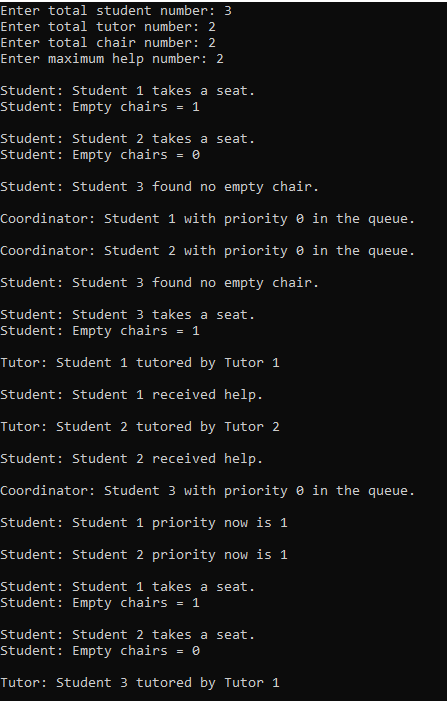
}

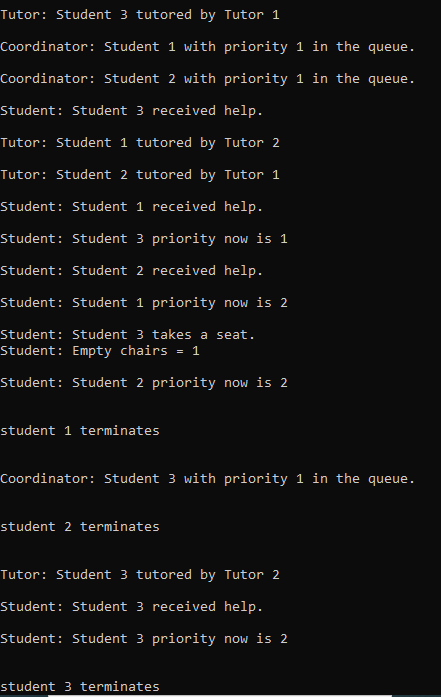
pthread\_join(coordinator, NULL);

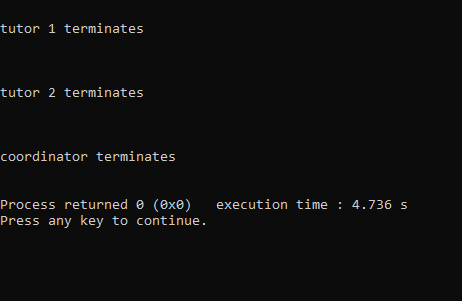
return 0;

}

**Output**

****





**Conclusion**

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