CSE316: OPERATING SYSTEM

Assignment – Simulation based

**Student name:** S. Dileep Sai Kumar

**Student ID:** 11701344

**Section no:** EE029

**Roll no: 14**

**Email address:** [dileepsingireddy88@gamil.com](mailto:dileepsingireddy88@gamil.com)

**GitHub link:** [**https://github.com/dileepsaikumar/dileepsaikumar**](https://github.com/dileepsaikumar/dileepsaikumar)

**Question: 14**

**PROBLEM:**

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 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 C and POSIX threads, mutex locks, and unnamed semaphores, implement a solution that coordinates the activities of the TA and the students. Details for this assignment are provided below.

**SOLUTION PROPOSED FOR THE PROBLEM:**

* Using Pthreads, n students are created. Each student, as well as the TA, run as a separate thread.
* Student threads alternate between programming for a period of time and seeking help from the TA. If the TA is available, they obtain help.
* Otherwise, they either sit in a chair in the hallway, or if no chairs are available, resume programming and seek help at a later time.
* If a student arrives and notices that the TA is sleeping, the student notifies the TA using a semaphore. When the TA finishes helping a student, the TA checks to see if there are students waiting for help in the hallway.
* If so, the TA helps each of these students in turn.
* If no students are present, the TA returns to napping.

**ALGORITHM:**

1) Create n students, each will run as a separate thread.

2) Student threads will alternate between programming for a period of time and seeking help from the TA.

3) If the TA is available, students will obtain help

4) If the TA is not available, students will sit in a chair in the hallway (waitList)

5) If no chairs are available, students will resume programming and seek help at a later time

6) If the TA is sleeping, the student will notify TA with a semaphore

7) Use a semaphore to indicate a student in help mode,

8) When a TA finishes helping a student, the TA must check to see if there are students waiting for help in the hallway.

9) The TA must help the students waiting in the hallway in the order that they started waiting.

10) If no students are waiting, then the TA can resume napping.

**COMPLEXITY:**

* O(N^2) Complexity.

**CODE: C CODE:**

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <semaphore.h>

void\* student\_actions( void\* student\_id );

void\* ta\_actions();

#define NUM\_WAITING\_CHAIRS 3

#define DEFAULT\_NUM\_STUDENTS 5

#include <stdio.h>

sem\_t sem\_students;

sem\_t sem\_ta;

pthread\_mutex\_t mutex\_thread;

int waiting\_room\_chairs[3];

int number\_students\_waiting = 0;

int next\_seating\_position = 0;

int next\_teaching\_position = 0;

int ta\_sleeping\_flag = 0;

int main( int argc, char \*\*argv )

{

int i;

int student\_num;

if (argc > 1 )

{

if ( isNumber( argv[1] ) == 1)

{

student\_num = atoi( argv[1] );

}

else

{

printf("Invalid input. Quitting program.");

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;

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\* ta\_actions()

{

printf( "Checking for students.\n" );

while( 1 )

{

//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;

sleep( help\_time );

pthread\_mutex\_unlock(&mutex\_thread );

sem\_post( &sem\_ta );

}

//if no students are waiting

else

{

if ( ta\_sleeping\_flag == 0 )

{

printf( "No students waiting. Sleeping.\n" );

ta\_sleeping\_flag = 1;

}

}

}

}

void\* student\_actions( void\* student\_id )

{

int id\_student = \*(int\*)student\_id;

while( 1 )

{

//if student is waiting, continue waiting

if ( isWaiting( id\_student ) == 1 )

{

continue;

}

//student is programming.

int time = rand() % 5;

printf( "\tStudent %d is programming for %d seconds.\n", id\_student, time );

sleep( time );

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 );

//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 );

}

}

}

int isNumber(char number[])

{

int i;

for ( i = 0 ; number[i] != 0; i++ )

{

if (!isdigit(number[i]))

return 0;

}

return 1;

}

int isWaiting( int student\_id ) {

int i;

for ( i = 0; i < 3; i++ ) {

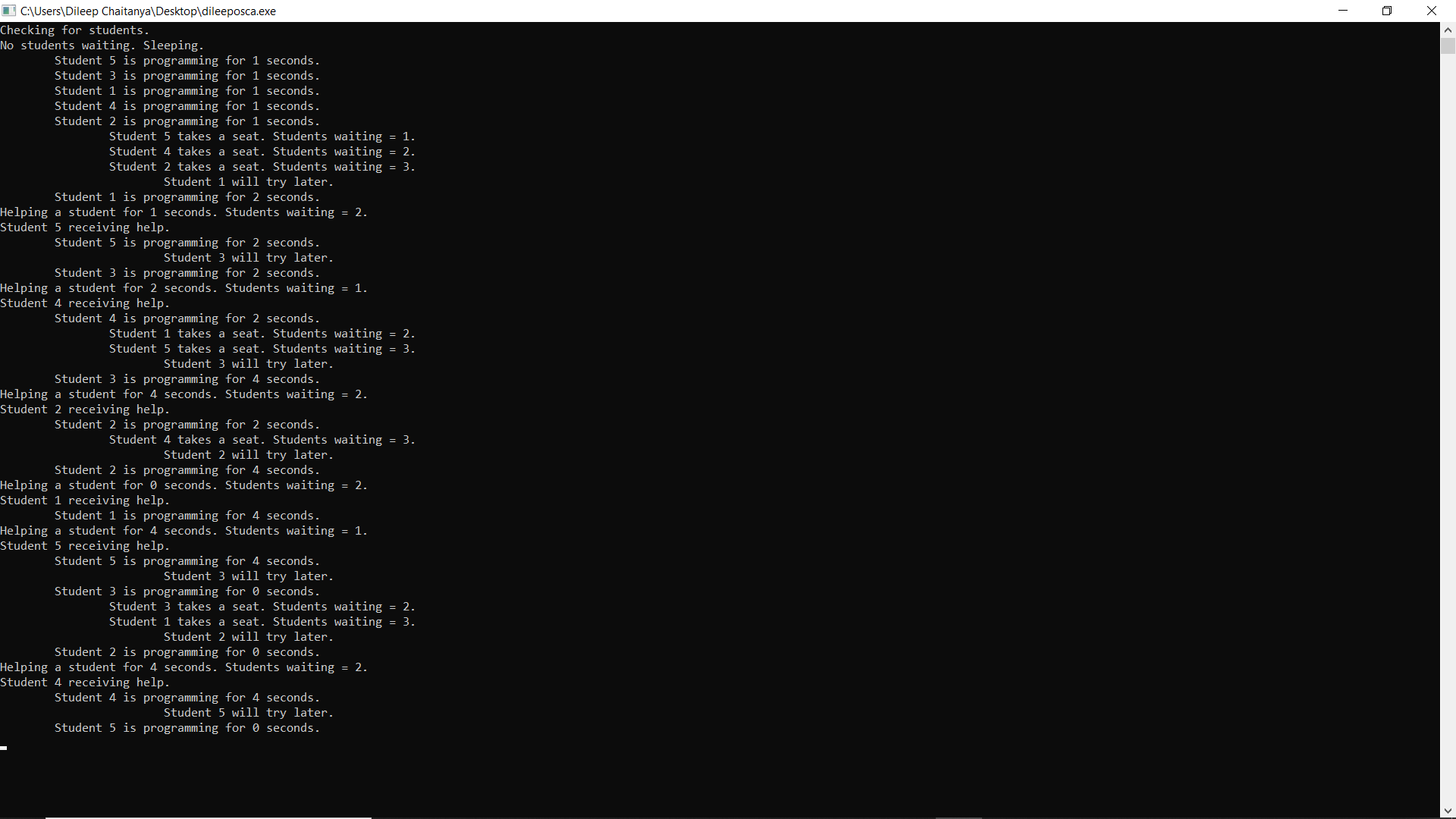
if ( waiting\_room\_chairs[i] == student\_id ) { return 1; }

}

return 0;

}

**SCREENSHOT FOR THE RUNNING CODE:**

****

**CONCLUSION:**

* So, far this problem we use C and POSIX threads, mutex locks, and unnamed semaphores to implement a solution that coordinates the activities of the TA and the students.
* The TA and each student thread print their state and threadID (student number).
* The program can take 0 or 1 command line parameters. The number of students’ present can be specified as a command line parameter. If this parameter is not included, the number of students defaults to 5.

**REFERENCES:**

* <https://www.alltestanswers.com/using-posix-threads-and-mutex-locks-and-semaphores-implement-a-solution-that-coordinates-the-activities-of-the-ta-and-the-students/>
* <https://github.com/dpulsifer/Sleeping-TA-Problem>