****

**PROJECT REPORT OF**

**OPERATING SYSTEMS LAB**

|  |  |
| --- | --- |
| **NAME:** | **Rabia**  **Zainab-Binte-Hassan** |
| **ID:** | **FA17-BECE-2001**  **FA17-BECE-0020** |
| **TEACHER NAME:** | **Ma’am Rafia** |
| **SECTION:** | **BE(CSE)--AM** |
| **PROJECT TOPIC:** | **Sleeping Teaching Assistant** |

**INTRODUCTION:**

This project deals with solving the Sleeping Teaching Assistant problem using thread synchronization. The implementation involves defining approaches to be taken for different scenarios as to how a Teaching Assistant (TA) and students interact with each other. The solution will be implemented using POSIX threads, mutex-locks, and semaphores for the different scenarios.

**PROJECT DESCRIPTION:**

The project problems are solved by the team using POSIX threads, semaphores and a FIFO queue in order to describe the behaviors of the TA and the students. Both the TA and students will have their own semaphores and separate p-threads. . By using a mutex-lock the situation of a deadlock can be avoided as either the TA or the student will have exclusive rights to change their state and this can be accomplished by only one person at a given instance. By using a First-In First-Out queue, starvation can be avoided as we know every student will be served according to the time they arrive at the TA’s office. Therefore, no student will be able to skip the line. The team approaches the Teaching Assistant Problem by solving 4 different scenarios. They are as follows:

1. Scenario One

There will be zero students coming to visit the TA and the TA will check the hallway outside his office to see if there are any students seated and waiting for him. If there are none, the TA will sleep in his office.

1. Scenario Two

When a student arrives at the TA’s office and finds the TA sleeping. Then the student will awaken the TA and ask for help. When the TA assists the student, the student's semaphore changes from 0 to 1 and waits for the TA's semaphore. When the TA finishes helping one student, he will check if there is any other student waiting in the hallway. If yes, he will help the next student and if not, TA goes back to sleeping and TA's semaphore becomes 1 and awaits student's semaphore.

1. Scenario Three

When a student arrives while the TA is busy with another student. Then the student who arrived will have to check if the TA is busy. If the TA is busy, the student will have to wait seated outside in the hallway until the TA is done with his session. When the TA completes his session, the student seated outside will be called in by the TA for a review session. Once all students have finished their sessions and left the TA’s office, the TA will go back to sleep after making sure no students are waiting.

1. Scenario Four

When a student arrives while the TA is busy in a review session, and all the seats in the hallway are occupied. Then, student will have to leave the hallway and come back later. When the student comes back, eventually, and there is a seat available, he will take a seat and wait for his turn with the TA.

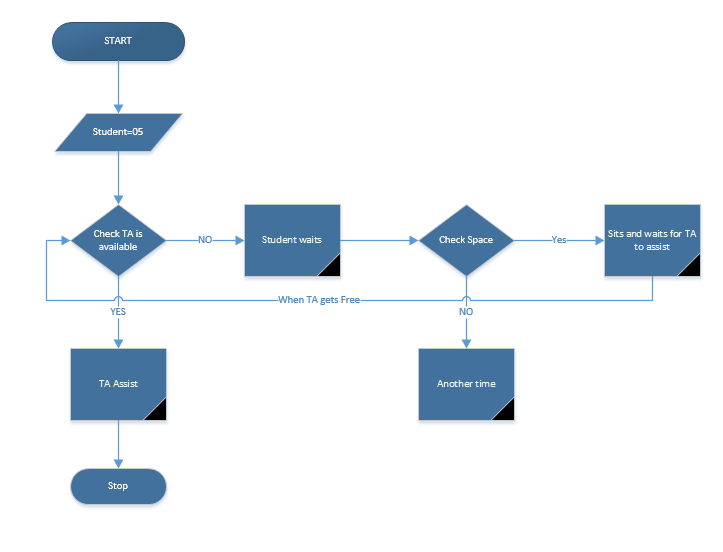
**ALGORITHM:**

* sem\_t sem\_stu - Student
* sem\_t sem\_ta - TA
* pthread\_mutex\_t mutex – Access Seats
* int count – Variable for number of waiting students

Initially, the semaphores and the variable are given the following initial values.

* Student = 0
* TA = 0
* Access Seats = 1
* Waiting students = 0

**FLOW CHART:**



**PROJECT CODE:**

#include <pthread.h> //Create POSIX threads.

#include <time.h> //Wait for a random time.

#include <unistd.h> //Thread calls sleep for specified number of seconds.

#include <semaphore.h> //To create semaphores

#include <stdlib.h>

#include <stdio.h> //Input Output

pthread\_t \*Students; //N threads running as Students.

pthread\_t TA; //Separate Thread for TA.

int ChairsCount = 0;

int CurrentIndex = 0;

//Declaration of Semaphores and Mutex Lock.

sem\_t TA\_Sleep;

sem\_t Student\_Sem;

sem\_t ChairsSem[3];

pthread\_mutex\_t ChairAccess;

//Declared Functions

void \*TA\_Activity();

void \*Student\_Activity(void \*threadID);

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

{

int number\_of\_students; //a variable taken from the user to create student threads. Default is 5 student threads.

int id;

srand(time(NULL));

//Initializing Mutex Lock and Semaphores.

sem\_init(&TA\_Sleep, 0, 0);

sem\_init(&Student\_Sem, 0, 0);

for(id = 0; id < 3; ++id) //Chairs array of 3 semaphores.

sem\_init(&ChairsSem[id], 0, 0);

pthread\_mutex\_init(&ChairAccess, NULL);

if(argc<2)

{

printf("Number of Students not specified. Using default (5) students.\n");

number\_of\_students = 5;

}

else

{

printf("Number of Students specified. Creating %d threads.\n", number\_of\_students);

number\_of\_students = atoi(argv[1]);

}

//Allocate memory for Students

Students = (pthread\_t\*) malloc(sizeof(pthread\_t)\*number\_of\_students);

//Creating TA thread and N Student threads.

pthread\_create(&TA, NULL, TA\_Activity, NULL);

for(id = 0; id < number\_of\_students; id++)

pthread\_create(&Students[id], NULL, Student\_Activity,(void\*) (long)id);

//Waiting for TA thread and N Student threads.

pthread\_join(TA, NULL);

for(id = 0; id < number\_of\_students; id++)

pthread\_join(Students[id], NULL);

//Free allocated memory

free(Students);

return 0;

}

void \*TA\_Activity()

{

while(1)

{

sem\_wait(&TA\_Sleep); //TA is currently sleeping.

printf("~~~~~~~~~~~~~~~~~~~~~TA has been awakened by a student.~~~~~~~~~~~~~~~~~~~~~\n");

while(1)

{

// lock

pthread\_mutex\_lock(&ChairAccess);

if(ChairsCount == 0)

{

//if chairs are empty, break the loop.

pthread\_mutex\_unlock(&ChairAccess);

break;

}

//TA gets next student on chair.

sem\_post(&ChairsSem[CurrentIndex]);

ChairsCount--;

printf("Student left his/her chair. Remaining Chairs %d\n", 3 - ChairsCount);

CurrentIndex = (CurrentIndex + 1) % 3;

pthread\_mutex\_unlock(&ChairAccess);

// unlock

printf("\t TA is currently helping the student.\n");

sleep(5);

sem\_post(&Student\_Sem);

usleep(1000);

}

}

}

void \*Student\_Activity(void \*threadID)

{

int ProgrammingTime;

while(1)

{

printf("Student %ld is doing programming assignment.\n", (long)threadID);

ProgrammingTime = rand() % 10 + 1;

sleep(ProgrammingTime); //Sleep for a random time period.

printf("Student %ld needs help from the TA\n", (long)threadID);

pthread\_mutex\_lock(&ChairAccess);

int count = ChairsCount;

pthread\_mutex\_unlock(&ChairAccess);

if(count < 3) //Student tried to sit on a chair.

{

if(count == 0) //If student sits on first empty chair, wake up the TA.

sem\_post(&TA\_Sleep);

else

printf("Student %ld sat on a chair waiting for the TA to finish. \n", (long)threadID);

// lock

pthread\_mutex\_lock(&ChairAccess);

int index = (CurrentIndex + ChairsCount) % 3;

ChairsCount++;

printf("Student sat on chair.Chairs Remaining: %d\n", 3 - ChairsCount);

pthread\_mutex\_unlock(&ChairAccess);

// unlock

sem\_wait(&ChairsSem[index]); //Student leaves his/her chair.

printf("\t Student %ld is getting help from the TA. \n", (long)threadID);

sem\_wait(&Student\_Sem); //Student waits to go next.

printf("Student %ld left TA room.\n",(long)threadID);

}

else

printf("Student %ld will return at another time. \n", (long)threadID);

//If student didn't find any chair to sit on.

}

}

**Reference Links:**

<https://>[github.com/shubhlpu16/Sleeping-Teaching-Assistant](https://github.com/shubhlpu16/Sleeping-Teaching-Assistant)

<https://www.chegg.com/homework-help/questions-and-answers/project-sleeping-teaching-assistant-university-computer-science-department-teaching-assist-q8730961>