**THE SLEEPING TEACHING ASSISTANT PROBLEM**

A

Mini Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

By

**BAKSHI ABHINITH**

**1602-19-733-124**

**AKASH S VORA**

**1602-19-733-126**

**KANCHARLA SRINAYAN**

**1602-19-733-171**



**Department of Computer Science & Engineering**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**2021**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Computer Science & Engineering**

****

**DECLARATION BY THE CANDIDATE**

I, **BAKSHI ABHINITH,** bearing hall ticket number, **1602-19-733-124**, hereby declare that the project report entitled **“THE SLEEPING TEACHING ASSISTANT PROBLEM”** Department of Computer Science & Engineering, VCE, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering**.

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

**Bakshi Abhinith,**

**1602-19-733-124.**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Computer Science & Engineering**

****

**BONAFIDE CERTIFICATE**

Thisis to certify that the project entitled **“THE SLEEPING TEACHING ASSISTANT PROBLEM”** being submitted by **BAKSHI ABHINITH,** bearing **1602-19-733-124,** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science & Engineering is a record of bonafide work carried out by him/her under my guidance.

**Ms. T. Nishitha**

**Assistant Professor**

**Dept. of CSE**

**ACKNOWLEDGEMENT**

With immense pleasure, we record our deep sense of gratitude to our guide Ms. T Nishitha, Assistant Professor, Vasavi College of Engineering, Hyderabad, for the valuable guidance and suggestions, keen interest and thorough encouragement extended throughout the period of the project work. I consider myself lucky enough to be part of this project. This project would add as an asset to my academic profile.

We express our thanks to all those who contributed for the successful completion of our project work.

**ABSTRACT**

The Teaching Assistant has to provide assistance to the under graduate students during office hours with their programming difficulties. The TA office is just a single room with a table and chair. It is not big enough to even accommodate two students at a time. While the TA is busy assisting a student inside office, the students were allowed to wait in the waiting area outside the office which accommodates three chairs. If a chair is empty, the student may occupy it and wait for his/her turn. However, if none of the chair is empty (i.e.) being occupied by some three students, the student decides to come back later. Each time, after assisting a student, TA checks with the students in the waiting area and calls them inside one by one to assist. TA will take a nap if he/she is left with no student in the waiting area. If incase a student arrives when TA is taking a nap, the student may wake up the TA asking for assistance.

**TABLE OF CONTENTS**

**Page No.**

1. **List of Figures …………………………………………………..1**
2. **Introduction……………………………………………………..2**
   1. **Overview…………………………………………………....3**
   2. **Concepts Used………………………………………………5**
   3. **Problem Statement…………………………………………8**
   4. **Objective……………………………………………………8**
3. **System Requirements .…………………....................................9**
4. **Implementation……..………………………………………….10**
5. **Output of the Program ………………………………………..20**
6. **Conclusions…………………….……………………………….25**
7. **References ………………………………………………...........25**

**LIST OF FIGURES**

Figure 2.1.1 Sleeping Teaching Assistant Problem…………………………….3

Figure 2.2.1 Semaphores………………………………………………………..5

Figure 2.2.2 Mutex Locks……………………………………………………….6

Figure 2.2.3 Threads…………………………………………………………….7

**INTRODUCTION**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), synchronization refers to one of two distinct but related concepts: synchronization of [processes](https://en.wikipedia.org/wiki/Process_(computer_science)), and synchronization of [data](https://en.wikipedia.org/wiki/Dataset). Process synchronization refers to the idea that multiple processes are to join up or [handshake](https://en.wikipedia.org/wiki/Handshaking) at a certain point, in order to reach an agreement or commit to a certain sequence of action. [Data synchronization](https://en.wikipedia.org/wiki/Data_synchronization) refers to the idea of keeping multiple copies of a dataset in coherence with one another, or to maintain [data integrity](https://en.wikipedia.org/wiki/Data_integrity). Process synchronization primitives are commonly used to implement data synchronization.

The Sleeping Teaching Assistant problem is one such process synchronization problem. The solution to this problem requires us to use concepts of semaphores, mutex locks and threads.

**2.1 OVERVIEW**

Three techniques were used to address this problem namely, a) semaphores b) mutual exclusion locks c) threads. We will be capturing two events for which two semaphores are used. One semaphore for the TA, and the other semaphore for students. We initialize 1 thread for TA, and N threads for N students. While the TA assists a student, other students are blocked and when the TA is done with assisting a student, TA will give preference to the student who is waiting for maximum time amongst all to assist next. If incase there were too many students to assist and the chairs in the waiting area were occupied, the student who has just entered the waiting room will leave and come at a later time to ask for TA’s help. On the other hand, TA will take a nap when there are no students to assist.

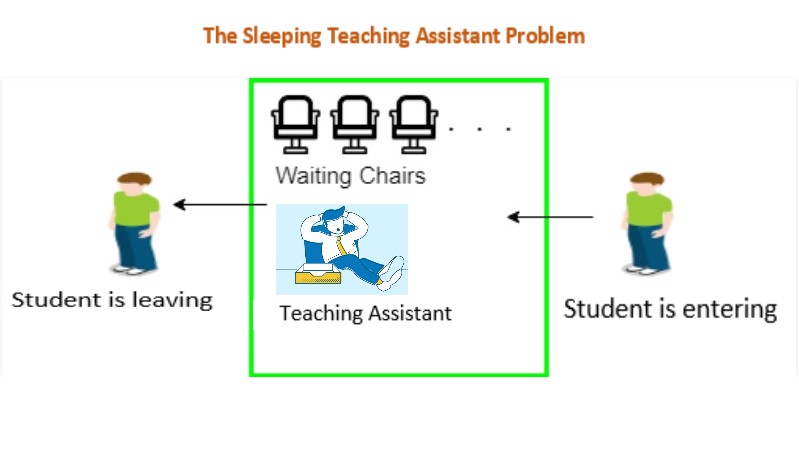


Figure 2.1.1

The TA has three actions. To check for a student, sleep if no one's waiting and wake up when a student arrives, we run a wait() operation on the Student\_Sem semaphore. When a student arrives, the TA starts assisting him/her, and then there is one less student in the waiting room. After the tutoring, the TA checks for a student and the cycle repeats.

The student also has three actions but, unlike the TA, the student only performs his/her actions once. When a student arrives, he/she checks to see if there's a chair available in the waiting room. If there isn't, the student leaves. Otherwise the student waits, and the number of students in the waiting room increases. The modification of the number of chairs is protected using mutex lock and unlock mechanisms.

* **POSSIBLE SCENARIOS :**
* **Scenario 1 :**

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

* **Scenario 2 :**

When a student arrives at the TA’s office and finds the TA sleeping. Then the the TA will be woken up the student and ask for help. When the TA finishes helping one student, he checks if there is any other student waiting in the hallway. If yes, he will help and if not ,TA goes back to sleeping.

* **Scenario 3 :**

When a student arrives while the TA is busy with another student then the student who arrived have to check if the TA is busy. If the TA is busy the student have to wait outside until the TA is done with his session. Once all students have finished their sessions and left the TA’s office the TA will go back to sleep.

**2.2 CONCEPTS USED**

1. **Semaphores :**

* A semaphore is a variable or abstract data type used to control access to a common resource by multiple processes and avoid critical section problems in a concurrent system such as a multitasking operating system. They are a useful tool in the prevention of race conditions. Semaphores which allow an arbitrary resource count are called counting semaphores, while those which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores and are used to implement locks.

In simple words, semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

The signal operation increments the value of its argument S.

The *<semaphore.h>*header defines the sem\_t type, used in performing semaphore operations. Some of the inbuilt functions in *<semaphore.h>* are sem\_init(), sem\_wait(), sem\_post() etc.,

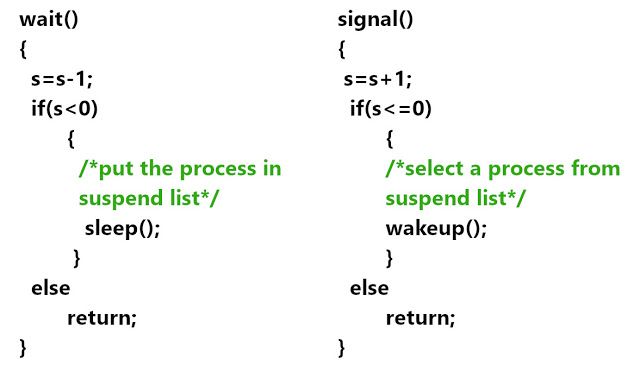


Figure 2.2.1

1. **Mutex Locks :**

* A Mutex is a lock that we set before using a shared resource and release after using it. When the lock is set, no other thread can access the locked region of code.

In simpler words, a mutex is a locking mechanism used to synchronize access to a resource. Only one task (can be a thread or process based on OS abstraction) can acquire the mutex. It means there is ownership associated with a mutex, and only the owner can release the lock (mutex). Some operations that can be performed on mutex locks in C are pthread\_mutex\_lock(), pthread\_mutex\_unlock() etc.,

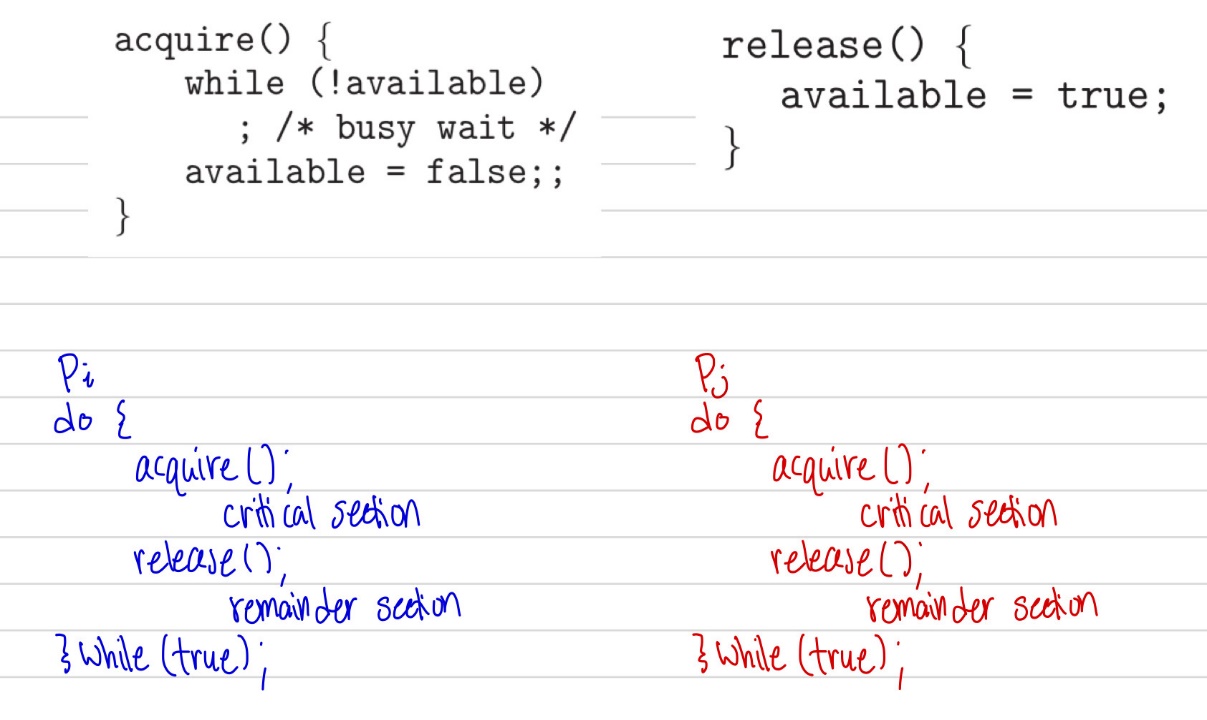


Figure 2.2.2

1. **Threads :**

* A thread is a path of execution within a process. A process can contain multiple threads. Threads are also known as lightweight processes. The idea is to achieve parallelism by dividing a process into multiple threads.

POSIX Threads, usually referred to as pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time.  *<pthread.h>* is a header file in C used to create POSIX threads.

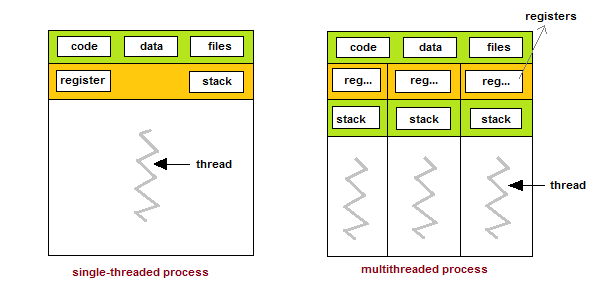
****

Figure 2.2.3

**2.3 PROBLEM STATEMENT**

Joy, a teaching assistant (TA) 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 3 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 wake up 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, then the student comes back at a later time.

**2.4 OBJECTIVE**

To solve the process synchronization problem – “Sleeping Teaching Assistant” using semaphores, mutex locks, and thread

**SYSTEM REQUIREMENTS**

**Hardware:**

* Minimum RAM required: 512 MB
* Input devices: Mouse, Keyboard
* Output devices: Monitor

**Software:**

* Online GDB Compiler
* Windows 7 or above

**IMPLEMENTATION**

//Sleeping Teaching Assistant Problem

#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>

#include <string.h>

const int number\_of\_chairs = 3;

//Declaration of Mutex Lock

pthread\_mutex\_t ChairAccess;

//Declaration of Semaphores

sem\_t TA\_Sem; //A semaphore to signal and wait TA's activities

sem\_t Student\_Sem; //A semaphore to signal and wait for Student's activities

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

pthread\_t TA; //A separate Thread for TA.

//Declaration of Functions

int isWaiting(int student\_id); //A function to check if the student is waiting in the hallway or not

void \*TA\_Activity(); //A function to run the activites of TA

void \*Student\_Activity(void \*threadID); //A function to run the activites of Students

int Waiting\_Room\_Chairs[3];

int Number\_Students\_Waiting = 0;

int Next\_Seating\_Position = 0;

int Next\_Teaching\_Position = 0;

int TA\_Sleep = 0;

void delay(float number\_of\_seconds)

{

int ms = 1000 \* number\_of\_seconds;

clock\_t start\_time = clock();

while (clock() < start\_time + ms);

}

int main()

{

int number\_of\_students;

//srand() is to prevent sequence repetition between runs

srand(time(NULL));

//Initializing Mutex Lock and Semaphores.

sem\_init(&TA\_Sem, 0, 1);

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

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

printf("Enter the number of students : ");

scanf("%d",&number\_of\_students);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

char \*welcome\_str = "\*\*\*\*\*\*\*\*\*\*\*WELCOME TO JOY'S PROGRAMMING CLASS\*\*\*\*\*\*\*\*\*\*\*";

for(int i=0;i<strlen(welcome\_str);i++)

{

printf("%c",welcome\_str[i]);

delay(1);

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("LET US SEE WHAT THE STUDENTS ARE DOING!!\n\n");

int student\_ids[number\_of\_students];

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

{

student\_ids[id] = id+1;

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

}

//Waiting for TA thread and N Student threads.

pthread\_join(TA, NULL);

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

{

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

}

free(Students);

return 0;

}

void \*TA\_Activity()

{

printf("Checking whether students are waiting for help......\n" );

while(1)

{

//Check if students are waiting

if(Number\_Students\_Waiting > 0 )

{

TA\_Sleep = 0;

sem\_wait(&Student\_Sem);

pthread\_mutex\_lock(&ChairAccess);

//CRITICAL SECTION BEGINS

int TA\_student\_teaching\_time = (rand()%5) + 1; //Bwtween 1 and 5 seconds

//TA helping the student with the programming assignment

printf("\nTA helping a student for %d seconds with the programming assignment.",TA\_student\_teaching\_time);

printf("\nStudents waiting to receive help = %d\n",(Number\_Students\_Waiting-1));

printf("Student %d receiving help from the TA.\n",Waiting\_Room\_Chairs[Next\_Teaching\_Position]);

Waiting\_Room\_Chairs[Next\_Teaching\_Position] = 0;

Number\_Students\_Waiting--;

Next\_Teaching\_Position = (Next\_Teaching\_Position + 1 ) % number\_of\_chairs;

sleep(TA\_student\_teaching\_time);

//CRITICAL SECTION ENDS

pthread\_mutex\_unlock(&ChairAccess);

sem\_post(&TA\_Sem);

}

//if none of the students are waiting for help

else

{

if(TA\_Sleep==0)

{

printf("No student is waiting in the hallway.\n\nTA is sleeping.....\n\n" );

TA\_Sleep = 1;

}

}

}

}

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

{

int ID\_Of\_Student = \*(int\*)threadID;

while(1)

{

//If student is waiting in the hallway, then he/she continues to wait

if (isWaiting(ID\_Of\_Student) == 1 )

{

continue;

}

//Now, the student is doing programming assignment.

int programming\_time = (rand() % 5) + 1;

printf("Student %d is programming for %d seconds.\n",ID\_Of\_Student, programming\_time);

sleep(programming\_time);

pthread\_mutex\_lock(&ChairAccess);

//CRITICAL SECTION BEGINS

if(Number\_Students\_Waiting < number\_of\_chairs)

{

Waiting\_Room\_Chairs[Next\_Seating\_Position] = ID\_Of\_Student;

Number\_Students\_Waiting++;

//Student takes a seat in the hallway

printf("Student %d takes a seat in the hallway.\nStudents waiting to receive help = %d\n",ID\_Of\_Student,Number\_Students\_Waiting);

Next\_Seating\_Position = (Next\_Seating\_Position + 1 ) % number\_of\_chairs;

//CRITICAL SECTION ENDS

pthread\_mutex\_unlock(&ChairAccess);

//Student needs to wake the TA if the TA is sleeping

sem\_post(&Student\_Sem);

sem\_wait(&TA\_Sem);

}

else

{

//If Number\_Students\_Waiting>= number\_of\_chairs, then this loop will be executed

//CRITICAL SECTION ENDS

pthread\_mutex\_unlock(&ChairAccess);

//No chairs available. Student will try later.

printf("Student %d will return at another time for help.\n",ID\_Of\_Student);

}

}

}

int isWaiting(int student\_id)

{

for(int i=0;i<number\_of\_chairs;i++)

{

if (Waiting\_Room\_Chairs[i]==student\_id )

{

return 1;

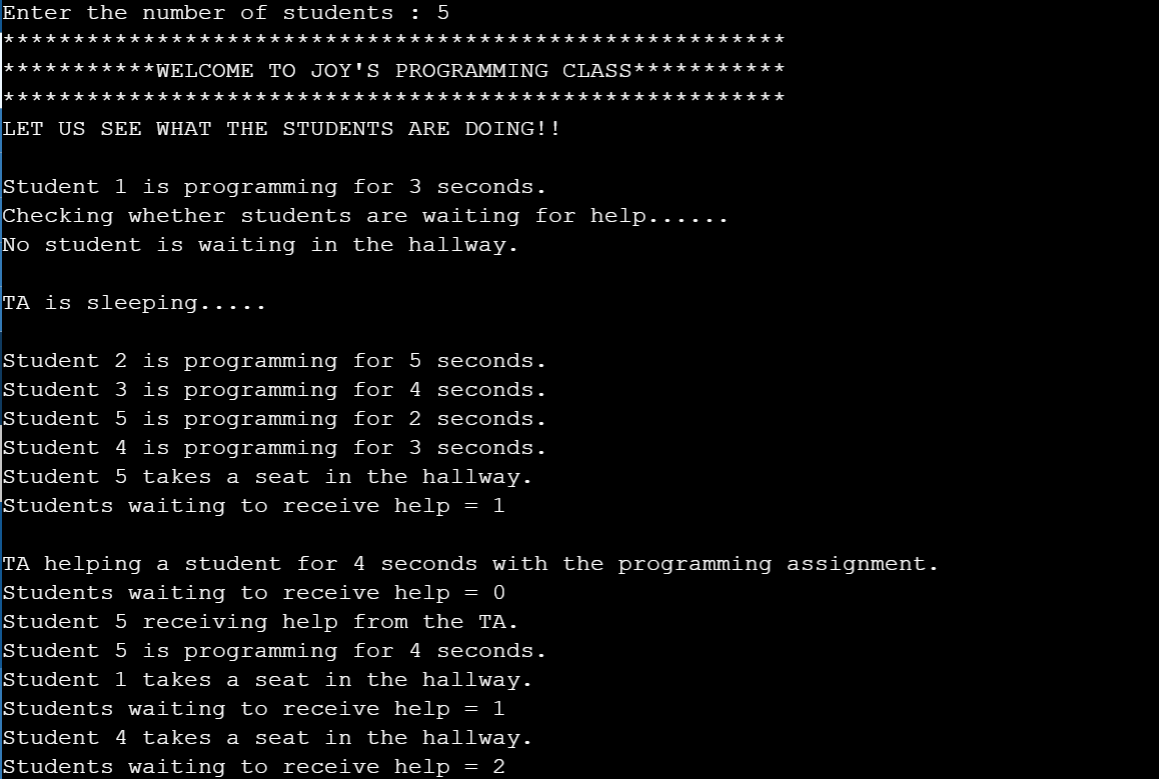
}

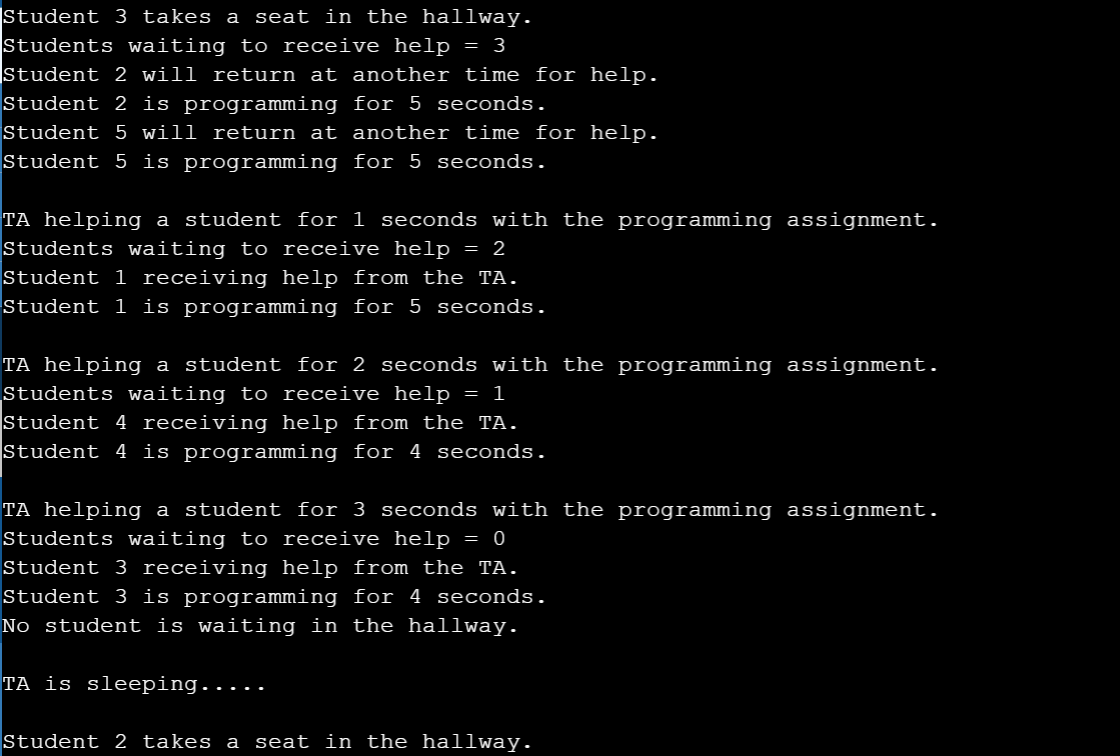
}

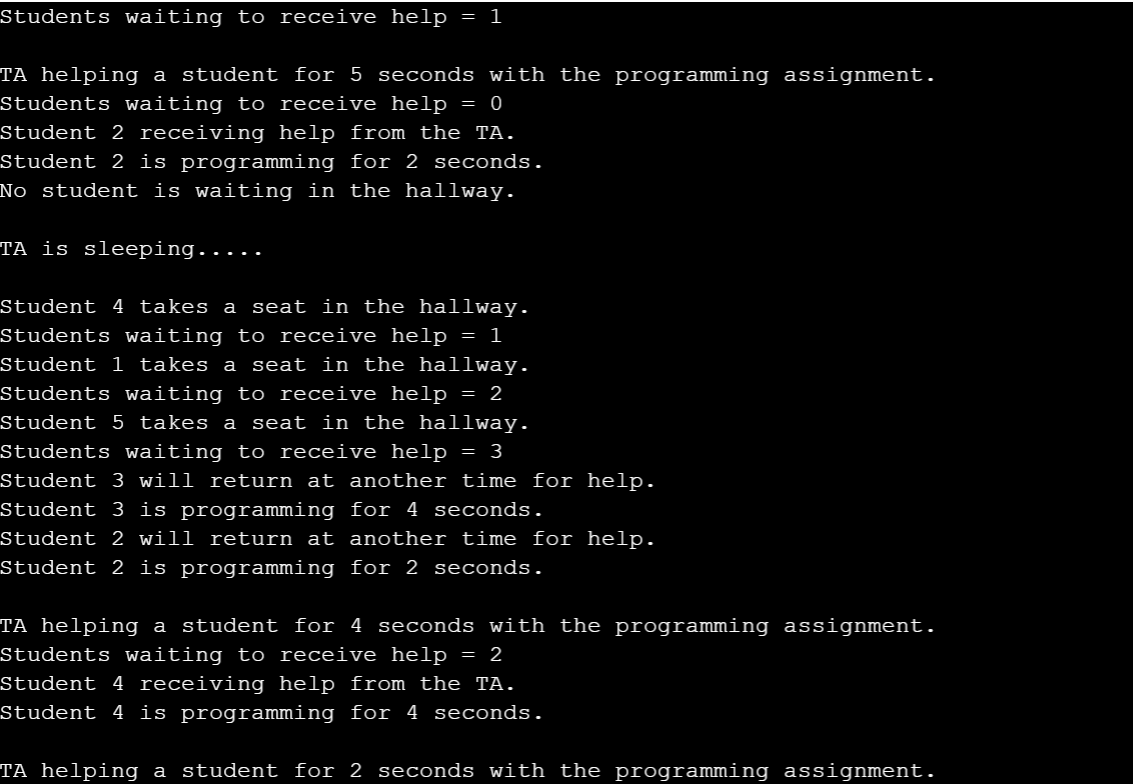
return 0;

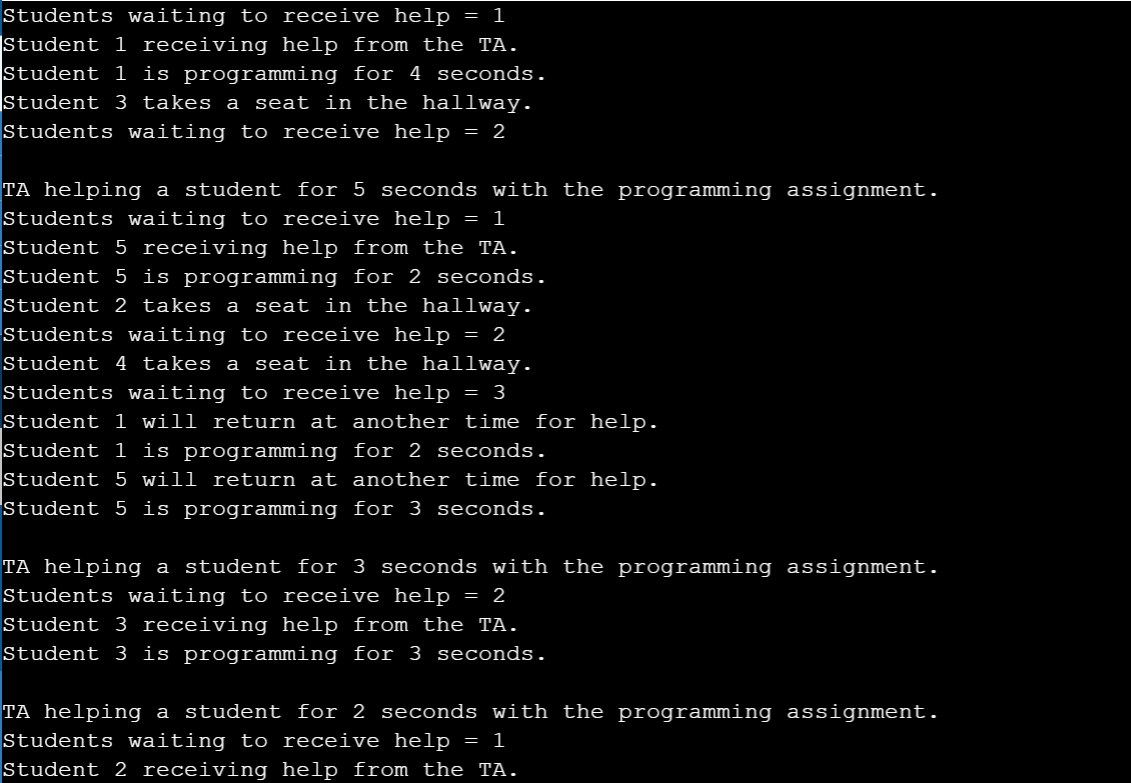
}

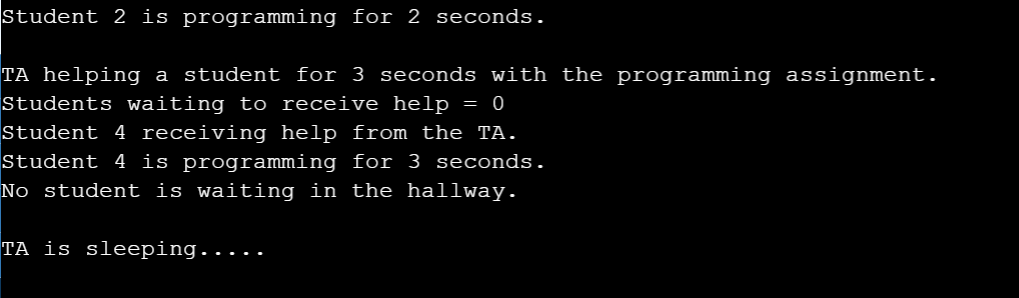
**OUTPUT OF THE PROGRAM**











**CONCLUSIONS**

If processes are not synchronized, it can lead to the inconsistency of shared data. So the change made by one process is not necessarily reflected when other processes access the same shared data. To avoid this type of inconsistency of data, the processes need to be synchronized with each other. In this Sleeping TA problem, N threads were used for N students. If these threads are not synchronized, it could have lead to a deadlock situation if threads simultaneously hold multiple locks. Thus, we have resolved all these issues in our program and successfully solved the process synchronization problem “Sleeping Teaching Assistant”.

**REFERENCES**

* Operating System Concepts (Ninth Edition) by Abraham Silberschatz, Peter Baer Galvin, Greg Gagne
* <https://www.geeksforgeeks.com>
* <https://stackoverflow.com/>
* <https://www.youtube.com>