

The American University in Cairo

Operating Systems

Dr Amr El-Kadi

Spring 19

The Sleeping Assistant

Chapter 7 – Project 2

Contents

Project Description According to Textbook	3
Assumptions.....	3
Procedures.....	4
Screenshots of the output.....	6
Limitations	8
Problems	8

Project Description According to Textbook

- 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 another randomized time.

Assumptions

- The TA does not have a finite number of helping units for the students. As long as the program is running this means these are the office hours of the TA. Which means that the program termination is not dependent on anything; it can run forever unless the user ends it by "ctrl+z" in the terminal. This is not analogous to real life; however, the concept of the behavior of the TA is.
- If the student arrives, there are three possibilities:

TA is asleep	Wake up the TA and get into his office chair
There is another student in the office and the chairs in the hallways aren't full	Wait in any available chair outside
There is another student in the office and the chairs are full	Wait a random time and come back later

- Students number is by default 5. Meaning that only IDs from 1 to 5 will be frequenting the TA's office. However; I also take it as an input before running the code.
- To run my code:

```
gcc -lpthread -pthread TA-Alaa.c -o TA
./TA "Number of students you specify"
```

Procedures

1. I created the needed threads, one for the TA and another is a thread pointer to the array of threads which is the students'.

```
void *StudentBehavior(void *threadID); //Passing thread ID to assign numbers to students
void *TABehavior();
```

I first started planning the architecture of my code and created the semaphores and mutexes that I will need:

```
sem_t SemTAAsleep;
sem_t SemStudent;
pthread_mutex_t VolatilityMutex;
```

The VolatilityMutex is used whenever there is a change in any volatile variable on any of the thread. It is a generic mutex.

2. I also created the Queue indices needed to treat students' arrival as if they arrived in a Queue (FIFO).
3. I created the student behavior function, which is the function passed to each thread of students. The function has three local variables:

<pre>int WaitRandTime;</pre>	To specify the random time the student wait till he/she asks for help from the TA (To make the intervals between the arrival of students randomized)
<pre>long SID = (long)threadID;</pre>	To store the thread ID to keep track of which student SID is currently running.
<pre>int localSlotCount = 0;</pre>	Count of taken chairs in the hallway of the TA's office.

-The students wait a random amount of time, when done, declare a student with SID needs help. Change localSlotsCount to the global count of slots.

If the localSlotsCount is more than the number of seats, print student will come at another time and do nothing. If it isn't, enqueue the student in the queue of seats.

Increment the number of slots occupied, let the Queue tail look at the next slot, let the local slots count be equal to the global count of occupied slots.

-If the local slots count is equal to 1 and the variable StudentInside is 0 (Which means there is no student inside and no students waiting outside; which means the TA must be asleep) ---> Wake the TA up. Else, wait for the TA.

Wait for the SemStudent to be signaled from the TA that the student is done with the help. After waiting, print that this student with this specific ID is done with the TA.

```
void *TABehavior()
```

4. The TA behavior function has two local variables:

```
int localSlotsCount = 0;
```

To store the global slots count and know if there is any student needing help

```
int localSID = 0;
```

To get the student's SID

-The TA wait the semaphore SemTAAsleep to be awakened by the first student or any student that would arrive in the middle while the TA is asleep. This is handled in the StudentBehavior function (If no students inside and slots count is 1, wake the TA).

-If TA is awakened, print that and lock the mutex over the slots to take their values.

Set the flag StudentInside to 1; which means there is a student inside.

Local slots count = global count.

Unlock the mutex.

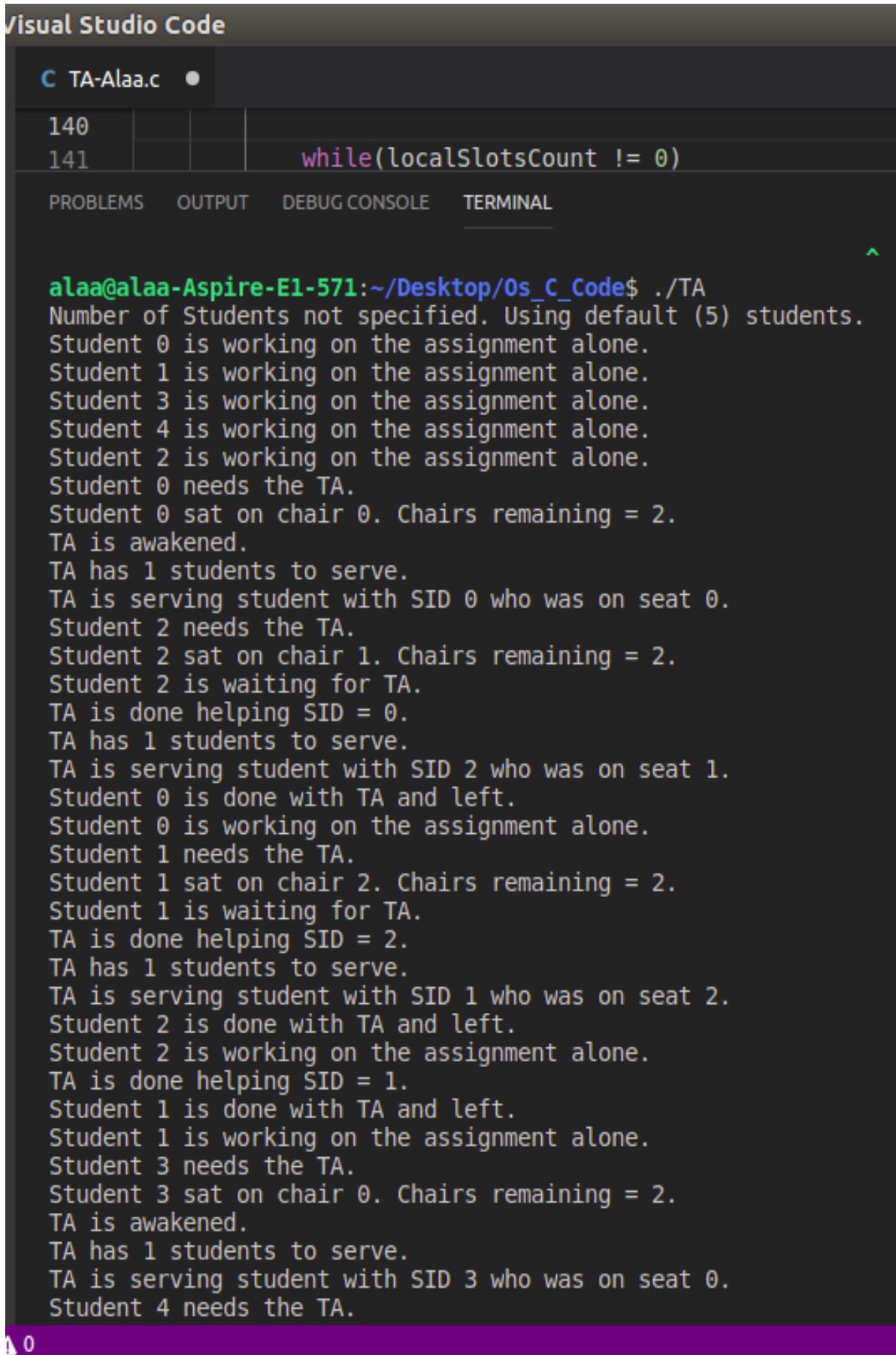
-If the local slots count is equal to 0, which is unreasonable and should not happen. This means that the student's threads awakened the TA while there was no slot occupied, print that this is an error "False alarm".

-After the TA is awake, stay awake till the serving of all students is done.

```
while(localSlotsCount != 0)
```

1. Lock volatility mutex
2. Print how many students you have to serve (Slots Occupied)
3. Dequeue the head of the queue to serve first student.
4. Print Seats(QueueHead) -> SID of the student to be served.
5. Decrease slots occupied.
6. Let the head of the queue point at next student to be served.
7. Unlock volatility mutex.
8. Sleep for a random time (Random helping time of the student we just dequeued)
9. Printf that you are done helping student with SID we got from Seats array.
10. Update the local slots count.
11. Do it again till no slot are occupied.
12. Go back to sleep if no more slots are occupied and send flag StudentInside = 0.

Screenshots of the output



```
Visual Studio Code
C TA-Alaa.c
140
141 while(localSlotsCount != 0)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

alaa@alaa-Aspire-E1-571:~/Desktop/0s_C_Code$ ./TA
Number of Students not specified. Using default (5) students.
Student 0 is working on the assignment alone.
Student 1 is working on the assignment alone.
Student 3 is working on the assignment alone.
Student 4 is working on the assignment alone.
Student 2 is working on the assignment alone.
Student 0 needs the TA.
Student 0 sat on chair 0. Chairs remaining = 2.
TA is awakened.
TA has 1 students to serve.
TA is serving student with SID 0 who was on seat 0.
Student 2 needs the TA.
Student 2 sat on chair 1. Chairs remaining = 2.
Student 2 is waiting for TA.
TA is done helping SID = 0.
TA has 1 students to serve.
TA is serving student with SID 2 who was on seat 1.
Student 0 is done with TA and left.
Student 0 is working on the assignment alone.
Student 1 needs the TA.
Student 1 sat on chair 2. Chairs remaining = 2.
Student 1 is waiting for TA.
TA is done helping SID = 2.
TA has 1 students to serve.
TA is serving student with SID 1 who was on seat 2.
Student 2 is done with TA and left.
Student 2 is working on the assignment alone.
TA is done helping SID = 1.
Student 1 is done with TA and left.
Student 1 is working on the assignment alone.
Student 3 needs the TA.
Student 3 sat on chair 0. Chairs remaining = 2.
TA is awakened.
TA has 1 students to serve.
TA is serving student with SID 3 who was on seat 0.
Student 4 needs the TA.
```

Visual Studio Code

C TA-Alaa.c ●

140

141

```
while(localSlotsCount != 0)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
Student 4 needs the TA.
Student 4 sat on chair 1. Chairs remaining = 2.
Student 4 is waiting for TA.
TA is done helping SID = 3.
TA has 1 students to serve.
TA is serving student with SID 4 who was on seat 1.
Student 3 is done with TA and left.
Student 3 is working on the assignment alone.
Student 0 needs the TA.
Student 0 sat on chair 2. Chairs remaining = 2.
Student 0 is waiting for TA.
TA is done helping SID = 4.
TA has 1 students to serve.
TA is serving student with SID 0 who was on seat 2.
Student 4 is done with TA and left.
Student 4 is working on the assignment alone.
Student 2 needs the TA.
Student 2 sat on chair 0. Chairs remaining = 2.
Student 2 is waiting for TA.
Student 1 needs the TA.
Student 1 sat on chair 1. Chairs remaining = 1.
Student 1 is waiting for TA.
TA is done helping SID = 0.
TA has 2 students to serve.
TA is serving student with SID 2 who was on seat 0.
Student 0 is done with TA and left.
Student 0 is working on the assignment alone.
TA is done helping SID = 2.
TA has 1 students to serve.
TA is serving student with SID 1 who was on seat 1.
Student 2 is done with TA and left.
Student 2 is working on the assignment alone.
Student 3 needs the TA.
Student 3 sat on chair 2. Chairs remaining = 2.
Student 3 is waiting for TA.
Student 4 needs the TA.
Student 4 sat on chair 0. Chairs remaining = 1.
Student 4 is waiting for TA.
```

Limitations

- The tracing of the errors was very hard.
- Understanding semaphores and what is needed exactly in the design of the solution of this project.
- Allowing for the finite resources concept in the project. I made running the program == TA has office hours.

Problems

1. Semaphores behavior:

I traced the semaphores behavior and whenever the TA is awoken from the students' behavior function the semaphore did not hold a value that I understood. That is why I added the extra variable to show me whether there is a student inside the TA's office and I reset the value of SemTAAsleep whenever the TA should go back to sleep again to 0.

2. The local variables had huge problems holding the most recent values.

I added one mutex for all volatile variable whenever they are accessed or update in threads.