

# Group Project 2: Application for Multithreaded Synchronization

*Antonio Hughes (015748783)*

*Sandra Chavez (016363540)*

*CECS 326 - Sec 04*

*Prof. Hailu Xu*

*10 - 18 - 2020*

## Abstract

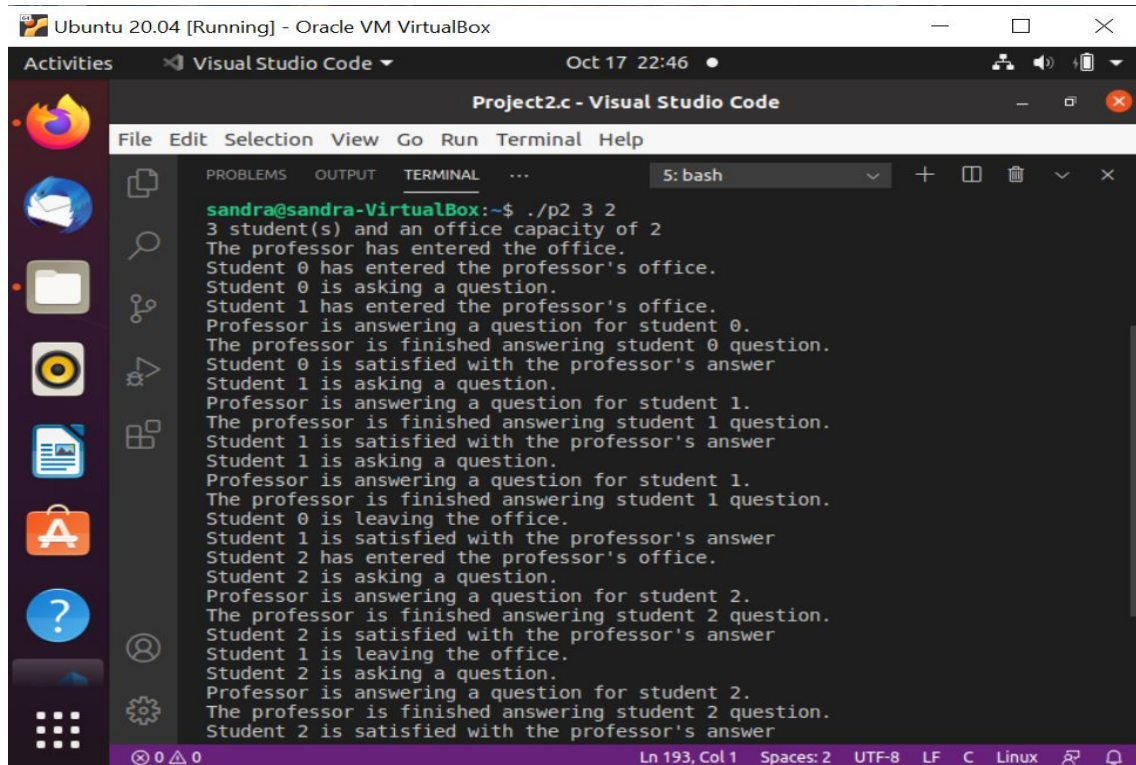
The second project focuses on topics utilized in the first project, but the difference is we are supposed to implement these topics in a real-world scenario. Similar to the first project, we are expected to use the threads programming interface: POSIX Threads (Pthreads). The real-world scenario/implementation is that we were hired by the CECS department at CSULB to write code to help synchronize a professor with their students during office hours. We are given a set of rules in the prompt that include: only one person can speak at a time, a student leaves the office once they are finished asking all their questions, etc. This project resembles the producer-consumer problem --in which both the producer and consumer share a common fixed-size buffer (office capacity).

The significance of the producer-consumer problem is having modern day operating systems focus on completing different processes. On a much smaller scale than what the whole operating system really does, we will focus on having a basic process finish in synchronization. The sole purpose of the producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is removing the data from the buffer one piece at a time. The main issue here is that we want to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. Failure to assess the producer-consumer problem correctly, will result in what is called a deadlock, which is a stranded hanging thread.

## Purpose

The purpose of this project is to demonstrate the implementation of multithreaded synchronization in the context of students going into a professor's office hours for questions. We will be able to set the student size, and office capacity size to whatever we want as long as it doesn't exceed the max thread size. With this, we can observe how the professor, who is consumer, handles answering students' questions as he follows through all the set number of students, which are the producers. We accomplish this by using mutexes, which allows us to synchronize and process threads by locking the data being stored, and semaphores, which allow us to control what data we are trying to access at any given time.

## Results



Ubuntu 20.04 [Running] - Oracle VM VirtualBox

Activities Visual Studio Code Oct 17 22:46

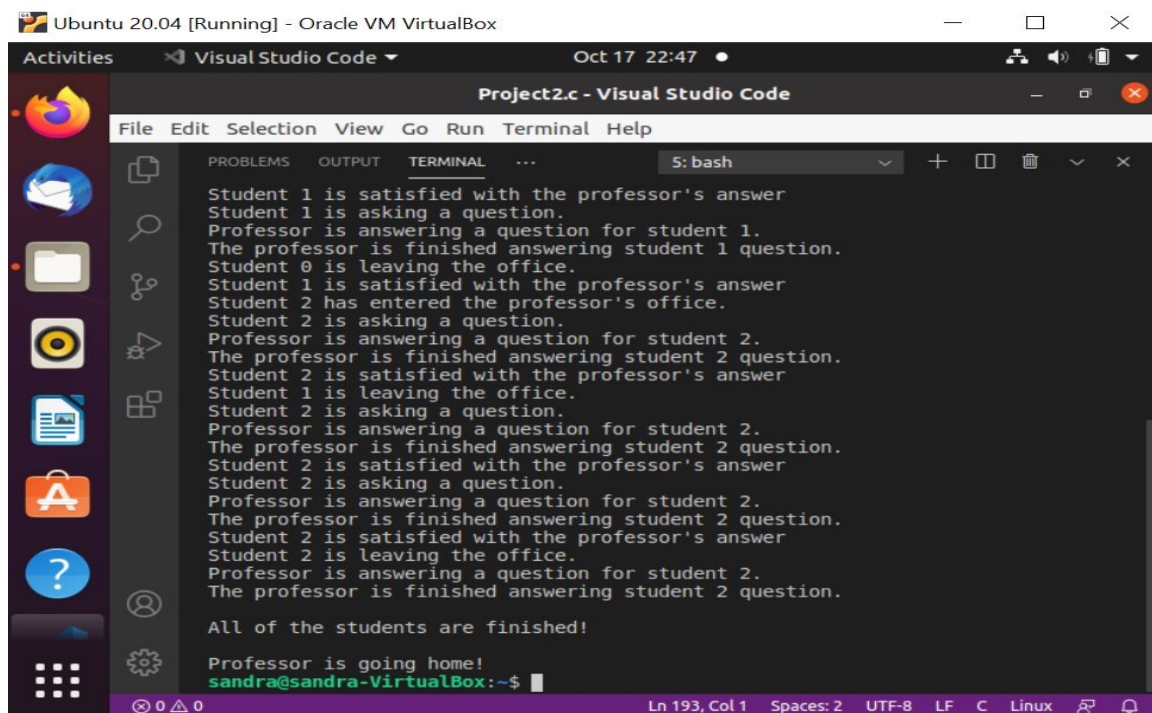
Project2.c - Visual Studio Code

File Edit Selection View Go Run Terminal Help

PROBLEMS OUTPUT TERMINAL ... 5: bash

```
sandra@sandra-VirtualBox:~$ ./p2 3 2
3 student(s) and an office capacity of 2
The professor has entered the office.
Student 0 has entered the professor's office.
Student 0 is asking a question.
Student 1 has entered the professor's office.
Professor is answering a question for student 0.
The professor is finished answering student 0 question.
Student 0 is satisfied with the professor's answer
Student 1 is asking a question.
Professor is answering a question for student 1.
The professor is finished answering student 1 question.
Student 1 is satisfied with the professor's answer
Student 1 is asking a question.
Professor is answering a question for student 1.
The professor is finished answering student 1 question.
Student 0 is leaving the office.
Student 1 is satisfied with the professor's answer
Student 2 has entered the professor's office.
Student 2 is asking a question.
Professor is answering a question for student 2.
The professor is finished answering student 2 question.
Student 2 is satisfied with the professor's answer
Student 1 is leaving the office.
Student 2 is asking a question.
Professor is answering a question for student 2.
The professor is finished answering student 2 question.
Student 2 is satisfied with the professor's answer
```

Ln 193, Col 1 Spaces: 2 UTF-8 LF C Linux



Ubuntu 20.04 [Running] - Oracle VM VirtualBox

Activities Visual Studio Code Oct 17 22:47

Project2.c - Visual Studio Code

File Edit Selection View Go Run Terminal Help

PROBLEMS OUTPUT TERMINAL ... 5: bash

```
Student 1 is satisfied with the professor's answer
Student 1 is asking a question.
Professor is answering a question for student 1.
The professor is finished answering student 1 question.
Student 0 is leaving the office.
Student 1 is satisfied with the professor's answer
Student 2 has entered the professor's office.
Student 2 is asking a question.
Professor is answering a question for student 2.
The professor is finished answering student 2 question.
Student 2 is satisfied with the professor's answer
Student 1 is leaving the office.
Student 2 is asking a question.
Professor is answering a question for student 2.
The professor is finished answering student 2 question.
Student 2 is satisfied with the professor's answer
Student 2 is asking a question.
Professor is answering a question for student 2.
The professor is finished answering student 2 question.
Student 2 is satisfied with the professor's answer
Student 2 is leaving the office.
Professor is answering a question for student 2.
The professor is finished answering student 2 question.

All of the students are finished!

Professor is going home!
sandra@sandra-VirtualBox:~$
```

Ln 193, Col 1 Spaces: 2 UTF-8 LF C Linux

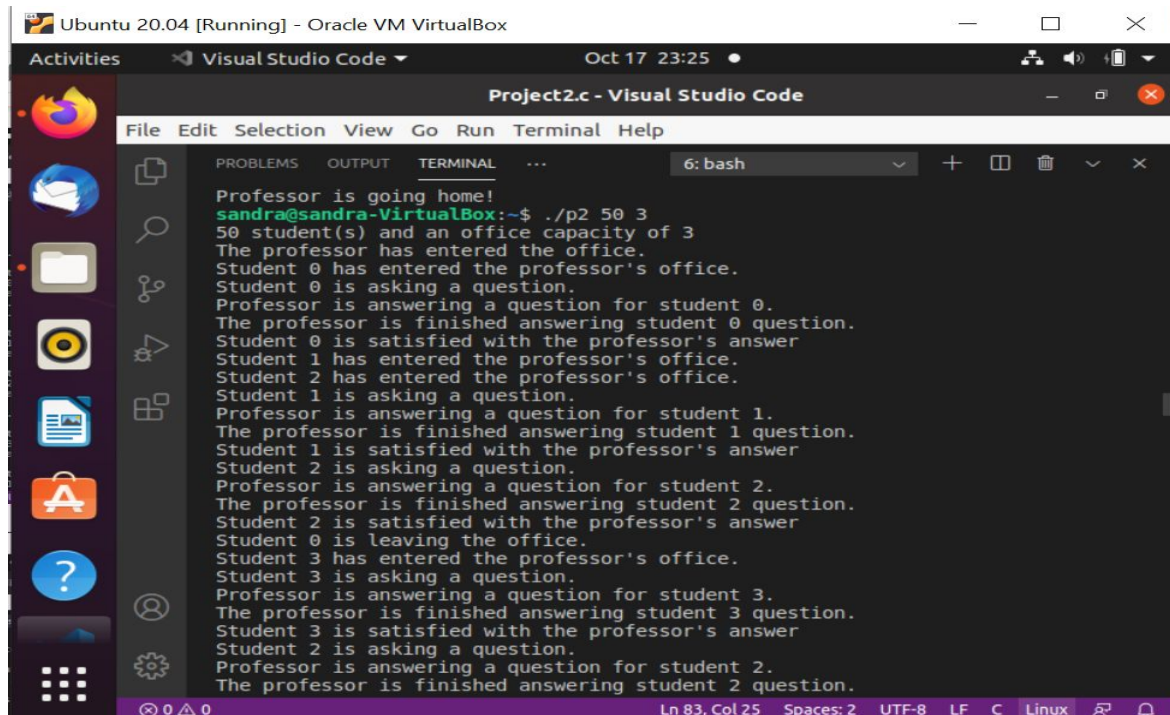
The screenshots posted above demonstrate the synchronization in the case of 3 students in an office with a capacity of two.

```
sandra@sandra-VirtualBox:~$ ./p2 30 3
30 student(s) and an office capacity of 3
The professor has entered the office.
Student 0 has entered the professor's office.
Student 0 is asking a question.
Professor is answering a question for student 0.
The professor is finished answering student 0 question.
Student 0 is satisfied with the professor's answer
Student 1 has entered the professor's office.
Student 2 has entered the professor's office.
Student 1 is asking a question.
Professor is answering a question for student 1.
The professor is finished answering student 1 question.
Student 1 is satisfied with the professor's answer
Student 2 is asking a question.
Professor is answering a question for student 2.
The professor is finished answering student 2 question.
Student 2 is satisfied with the professor's answer
Student 0 is leaving the office.
Student 3 has entered the professor's office.
Student 3 is asking a question.
Professor is answering a question for student 3.
The professor is finished answering student 3 question.
Student 3 is satisfied with the professor's answer
Student 1 is asking a question.
Professor is answering a question for student 1.
The professor is finished answering student 1 question.
```

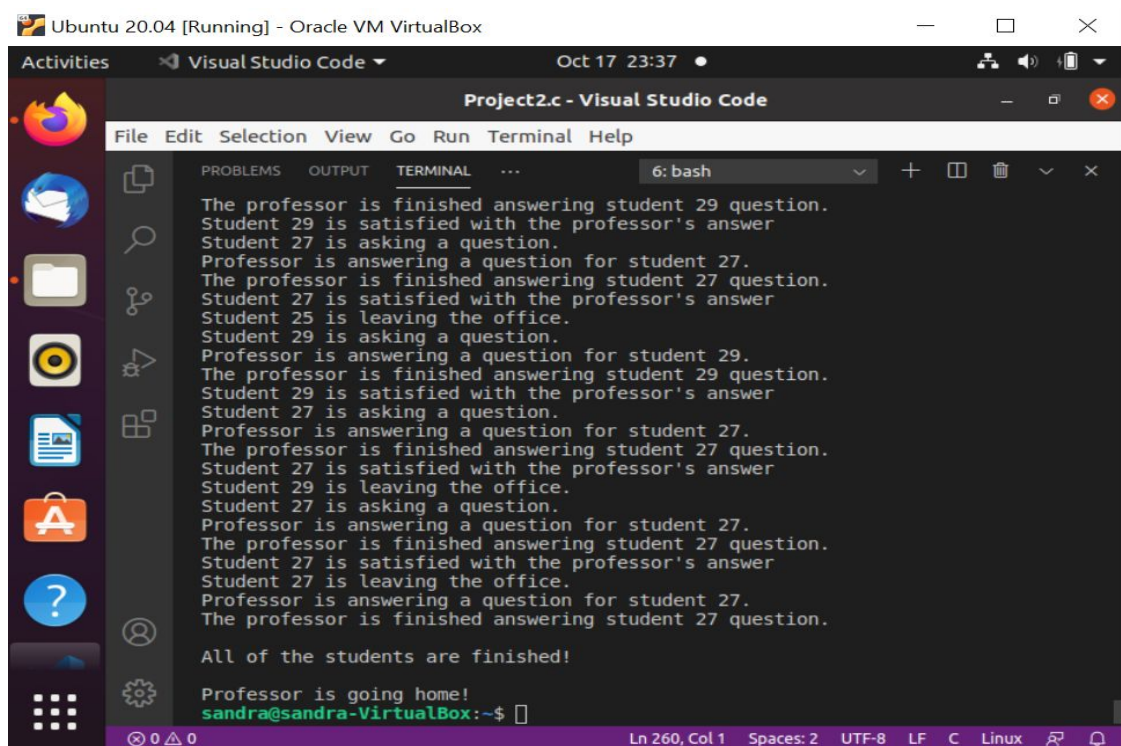
```
Student 26 is leaving the office.
Student 29 has entered the professor's office.
Student 29 is asking a question.
Professor is answering a question for student 29.
The professor is finished answering student 29 question.
Student 28 is leaving the office.
Student 28 is satisfied with the professor's answer
Student 27 is asking a question.
Professor is answering a question for student 27.
The professor is finished answering student 27 question.
Student 27 is satisfied with the professor's answer
Student 27 is asking a question.
Professor is answering a question for student 27.
The professor is finished answering student 27 question.
Student 27 is satisfied with the professor's answer
Student 29 is asking a question.
Professor is answering a question for student 29.
The professor is finished answering student 29 question.
Student 29 is satisfied with the professor's answer
Student 27 is leaving the office.
Student 29 is leaving the office.
Professor is answering a question for student 29.
The professor is finished answering student 29 question.
All of the students are finished!
Professor is going home!
sandra@sandra-VirtualBox:~$
```

The screenshots posted above are with the number of students increased to 30 and the office capacity the same.





```
Professor is going home!  
sandra@sandra-VirtualBox:~$ ./p2 50 3  
50 student(s) and an office capacity of 3  
The professor has entered the office.  
Student 0 has entered the professor's office.  
Student 0 is asking a question.  
Professor is answering a question for student 0.  
The professor is finished answering student 0 question.  
Student 0 is satisfied with the professor's answer  
Student 1 has entered the professor's office.  
Student 2 has entered the professor's office.  
Student 1 is asking a question.  
Professor is answering a question for student 1.  
The professor is finished answering student 1 question.  
Student 1 is satisfied with the professor's answer  
Student 2 is asking a question.  
Professor is answering a question for student 2.  
The professor is finished answering student 2 question.  
Student 2 is satisfied with the professor's answer  
Student 0 is leaving the office.  
Student 3 has entered the professor's office.  
Student 3 is asking a question.  
Professor is answering a question for student 3.  
The professor is finished answering student 3 question.  
Student 3 is satisfied with the professor's answer  
Student 2 is asking a question.  
Professor is answering a question for student 2.  
The professor is finished answering student 2 question.
```



```
The professor is finished answering student 29 question.  
Student 29 is satisfied with the professor's answer  
Student 27 is asking a question.  
Professor is answering a question for student 27.  
The professor is finished answering student 27 question.  
Student 27 is satisfied with the professor's answer  
Student 25 is leaving the office.  
Student 29 is asking a question.  
Professor is answering a question for student 29.  
The professor is finished answering student 29 question.  
Student 29 is satisfied with the professor's answer  
Student 27 is asking a question.  
Professor is answering a question for student 27.  
The professor is finished answering student 27 question.  
Student 27 is satisfied with the professor's answer  
Student 29 is leaving the office.  
Student 27 is asking a question.  
Professor is answering a question for student 27.  
The professor is finished answering student 27 question.  
Student 27 is satisfied with the professor's answer  
Student 27 is leaving the office.  
Professor is answering a question for student 27.  
The professor is finished answering student 27 question.  
  
All of the students are finished!  
  
Professor is going home!  
sandra@sandra-VirtualBox:~$
```

The screenshots posted above are with the number of students further increased to 50 and the capacity staying the same.

### **Contributions by group members:**

- **Antonio Hughes**
  - *Focused on the Professor function*
  - *Debugging the wait between the questions answered*
  - *Worked on the report*
- **Sandra Chavez**
  - *Focused on the Student function*
  - *Debugging the Student function*
  - *Made the makefile*
  - *Made the readme file*
  - *Worked on the report*

## Discussion/Analysis

For our analysis, we were able to note several things. First off, we had a lot of errors in our code in how the mutexs were processing the different student threads throughout our program. This made it really hard to see the expected results of having the professor answer all of the students' questions in time. Another major issue we had in our work was deadlock, or a hanging thread. This is caused by a set of processes trying to acquire a resource from one another however, there is miscommunication among the processes and now these resources are blocked from each other. This is a common issue to face when solving the producer-consumer problem. Once we were able to solve what was causing this, we were able to observe how the mutexes' block and flow the movement of our data for our student threads. Handling processes one by one to observe the flow of resources in an operating system such as Linux, allowed us to take a look at an everyday occurrence on all of our machines. Our results were as we predicted it to be and now have a solid understanding of multithreaded synchronization.