Min Kyaw - Section 05 - 018182136

Donovan Lee - Section 06 - 016741645

Lab 2 Report

In the header, we have many libraries included in order for the code to work. We have six mutex variables - office_mutex, leave_mutex, question_mutex, answer_mutex, professor_wake, and bool_lock. We have four pthread condition variables - quest_cond, answer_cond, office_cond, and professor_wait. We have four global integer variables and a static thread integer variable, and three booleans to later be used in our methods. Then we had to initialize our functions so that the compiler could know where the functions are being called from.

In our **AnswerStart()** function, we lock the answer_start mutex variable and go into a while loop involving the boolean question variable, and a condition wait is invoked involving the quest_cond, and answer_mutex variable which is to wait before the professor starts to answer the question. In our **AnswerDone()** function, we print out that the professor is done answering the question and have a lock on the mutex variable bool_lock to make sure that the professor is done answering the question. Question is set to false and the answer is set to true, and the bool_lock is unlocked and the condition signal is invoked to the answer_cond to also make sure that the answer is finished. Finally, the answer_mutex is unlocked.

In our **QuestionStart()** function, we have a mutex lock on the question_mutex and bool_lock to ensure that no other threads are interrupting when a student is asking a question. While it's being locked, we set our question variable to true and answer variable to false. We then unlock the bool_lock and signal the quest_cond condition to start. We then print that a student asks a question and go into a while loop involving the answer, printing out the fact that the student is waiting for an answer, then a wait condition is invoked involving the answer_cond and question_mutex. In our **QuestionDone()** function, we print out that the student is satisfied and finally unlock the question_mutex since the question is done.

In our **main**() function, we initialized our pthread mutexes and pthread condition functions to be used throughout our program. We verified our arguments that are passed through the command line, if they are not what we expect then the program ends. We created two pthreads, one for students and one for the professor. After the program finishes, all pthreads are destroyed, such as mutexes, conditions, and the thread itself.

The **professor**() function only does one thing, it loops **AnswerStart**() and **AnswerDone**() until all threads are done. Then the pthread exits as all the student threads are finished.

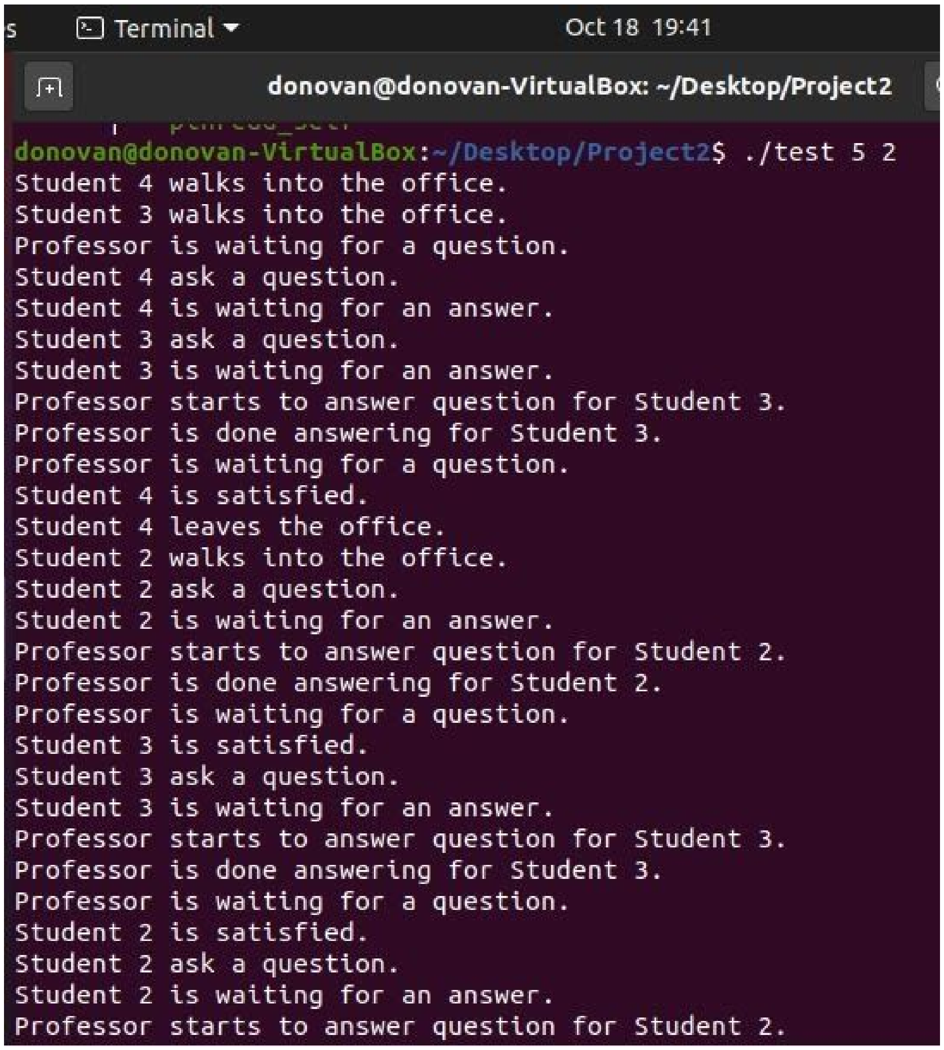
In our ***StudentID**() function we get multiple pthread ids that are set to a global variable so that it could be used for functions that depend on global variables such as pthreads. Then we use the student ids to create questions to be used in a for loop for **QuestionsStart**() and **QuestionsDone**(). After the loop finishes we call the **LeaveOffice**() function that decrements the occupancy in the office so a new thread/student could enter. After that the pthread can exit and a new thread can begin.

The **LeaveOffice**() function decrements the occupancy so that more students could enter the office for help. It has a mutex lock on it so only one student could leave at a time so that not everyone leaves at the same time. Then it signals to the wait condition in **EnterOffice**() so that the next student can enter.

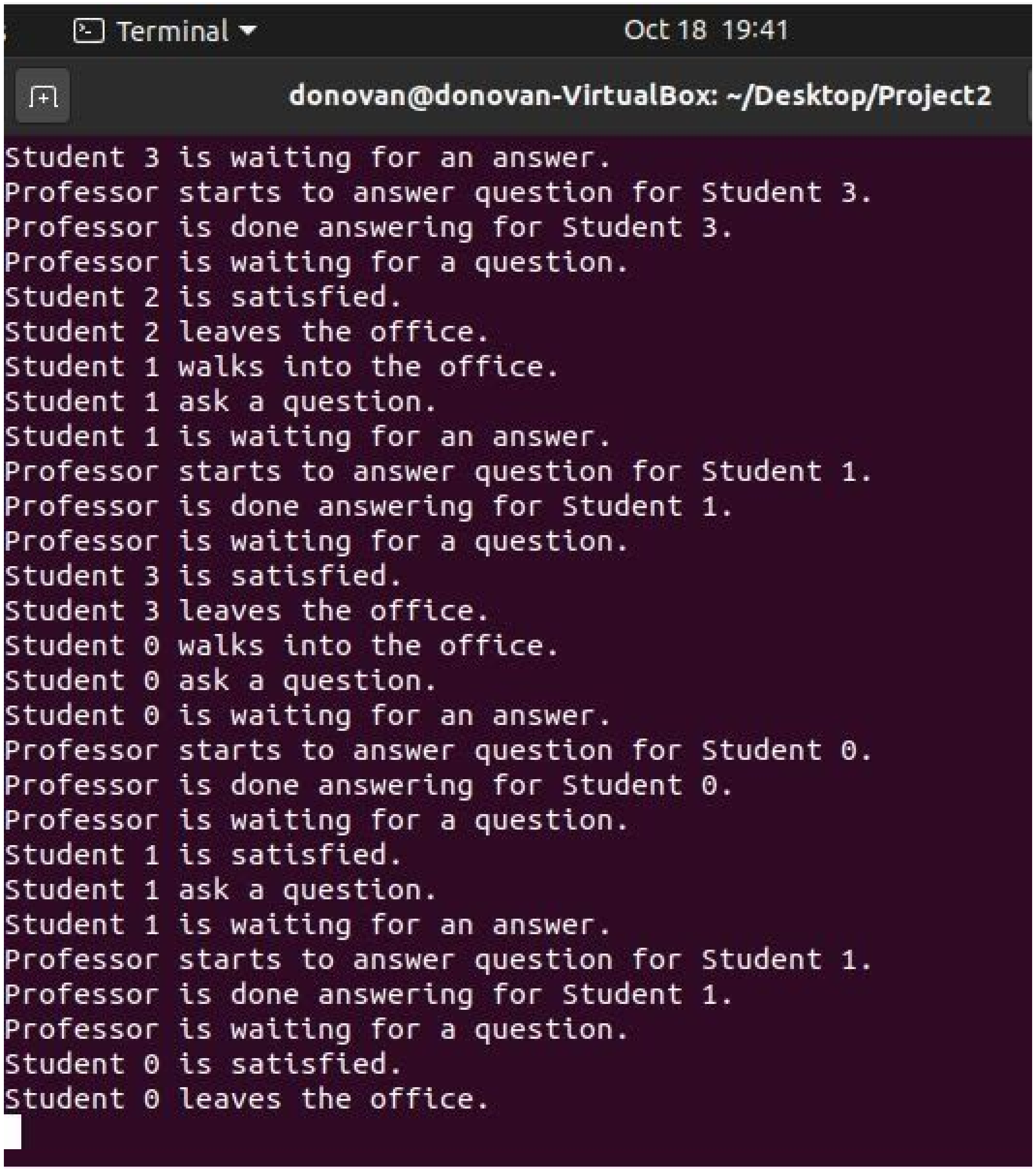
The **EnterOffice**() function takes in a student. The function is first locked so that one thread could enter one at a time. There's a while loop that checks to see if the room is full, if it is full then a print statement will show who is in the waiting room. Then there would be a wait condition in the while loop that would need to be signaled in the **LeaveOffice**() to let a waiting student in. After that occupancy would increase, and the lock mutex would unlock so the next student could come through and see if there is space available in the office.

Our code partially works for some reason. All our functions were completely finished but for some apparent reason the logic of how the functions run don’t correlate to how the output should be printed. When the project is ran the first thing the user should see that is printed is a certain amount of student #s being printed due to the capacity of the office. Then the next step would be the student asking the question. This is where the problem starts. The first student would ask a question then another student asks a question, but only one student should be asking a question at a time instead of multiple students. Then the output would also be wrong because the student who asked the question would not be synced with the professor giving the answer. That only happens for the first student that enters after that every student number is synced correctly. Then another problem is that our professor prints are too slow for some reason because a student would be satisfied and leave but the professor would answer the question later on. Due to that we got an infinite runtime where our program would not end.

Code Compiled


Terminal
Oct 18 19:41
donovan@donovan-VirtualBox: ~/Desktop/Project2

```
donovan@donovan-VirtualBox:~/Desktop/Project2$ ./test 5 2
Student 4 walks into the office.
Student 3 walks into the office.
Professor is waiting for a question.
Student 4 ask a question.
Student 4 is waiting for an answer.
Student 3 ask a question.
Student 3 is waiting for an answer.
Professor starts to answer question for Student 3.
Professor is done answering for Student 3.
Professor is waiting for a question.
Student 4 is satisfied.
Student 4 leaves the office.
Student 2 walks into the office.
Student 2 ask a question.
Student 2 is waiting for an answer.
Professor starts to answer question for Student 2.
Professor is done answering for Student 2.
Professor is waiting for a question.
Student 3 is satisfied.
Student 3 ask a question.
Student 3 is waiting for an answer.
Professor starts to answer question for Student 3.
Professor is done answering for Student 3.
Professor is waiting for a question.
Student 2 is satisfied.
Student 2 ask a question.
Student 2 is waiting for an answer.
Professor starts to answer question for Student 2.
```



```
Student 3 is waiting for an answer.
Professor starts to answer question for Student 3.
Professor is done answering for Student 3.
Professor is waiting for a question.
Student 2 is satisfied.
Student 2 leaves the office.
Student 1 walks into the office.
Student 1 ask a question.
Student 1 is waiting for an answer.
Professor starts to answer question for Student 1.
Professor is done answering for Student 1.
Professor is waiting for a question.
Student 3 is satisfied.
Student 3 leaves the office.
Student 0 walks into the office.
Student 0 ask a question.
Student 0 is waiting for an answer.
Professor starts to answer question for Student 0.
Professor is done answering for Student 0.
Professor is waiting for a question.
Student 1 is satisfied.
Student 1 ask a question.
Student 1 is waiting for an answer.
Professor starts to answer question for Student 1.
Professor is done answering for Student 1.
Professor is waiting for a question.
Student 0 is satisfied.
Student 0 leaves the office.
```

Contributions:

Donovan: Typed up the code

Minh: helped with the logic of the code and set up the Lab report