Can BEYAZNAR
161044038

CSE344 MIDTERM REPORT

1) PROBLEM SOLUTION

In this assignment, I first checked the input from the user. The input from the user is taken as in the bonus section. Then I identified the semaphores needed to solve the problem. And I defined shared memory. In the program, there should be a total of N + M + 1 processes. In order for these processes to function properly, I keep the IDs of my processes in an array. So I know which processes the supplier, cooks and students are involved in. I will explain how I have solved it by dividing it into the Supplier, cook and student sections.

A) Supplier Part

The purpose of the supplier is to take the food to the kitchen. It sends by selecting the type of food according to the correspondence of the letters read from the file. The common place of the supplier and cooks is the kitchen. Therefore, the cook should not enter while the supplier leaves food in the kitchen. I used mutex to achieve this. The supplier reads a letter from the file and locks the kitchen with sem_wait before sending the corresponding food. And after doing the necessary actions, I unlock with sem_post. Supplier performs this operation 3 * L * M times. The messages that need to be written during this loop are written with the write() function. If there is any character other than P, C and D in the file, the program ends with an error message. I also use 3 semaphores to count the dishes sent by the supplier. Also, thanks to the semaphore, I check for free space in the kitchen. Thus, if the kitchen is full, the supplier is waiting for free space. I use these semaphores in other parts of the program. To solve this problem, I took advantage of the producer consumer problem described in the lesson.

B) Cooks Part

The Cooks part is the longest part of the assignment. First of all, I used nested loop in this section. The first of these loops continues until the supplier sends all the dishes. The loop in it continues until the meals in the kitchen are over. In this part, I used mutex in the kitchen section in order to get the dishes sent by the supplier without any problems. While a cook bought a plate from the kitchen, I used another mutex to prevent other cooks from entering the kitchen. Thanks to this mutex, my program will work safely. And finally, I ensure that the program works safely by using the mutex which is for counter part, since the chefs should send food to the counter. Thus, the students are waiting while the cooks send the food to the counter. I applied a simple algorithm to select the food that should be sent to the counter. First of all, thanks to the semaphores, I get the number of soups, main dishes and desserts on the counter. And I'm sending the least food on the counter. Of course, while doing these operations, I check whether there is free space on the bench. To solve this problem, I took advantage of the producer consumer problem described in the lesson. I was encountering a minor problem in the cook section. While the chefs put the plates on the counter, in some cases, they did not send the last plate to the counter. And it was deadlock. To prevent this, I have added a few conditions to my code. If the supplier sent all the dishes, and only one plate left in the kitchen, send it to the counter and end the loops. While sending to the counter, I check if there is a free space in the counter.

C) Student Part

First of all, there is one loop in my student section. In this loop, I first look at which student is working. If this is a graduate student, he takes food from the counter before other students. I look at this control from the array where I hold my process IDs. If the currently running process tag ID is less than G (pidIndex <G), provide this priority. If not, it takes whenever it can take because it is an "undergraduate student". Then I print the messages the user wants with the write () function. And the student takes the dishes from the counter with sem_wait (). After taking these 3 plates, 3 places are emptied from the counter with the help of sem_post. After receiving the plates, the student checks to see if there is an empty table. And if any, the student writes a message to the terminal and

continues. This process runs L times for each student. I encountered a problem while doing this part. The cycle continues until the limitOfStudent <= L. However, when there is no food on the counter, the student cannot get food, but the value of the limitOfStudent parameter increased. So I added a small condition to my code. If there is no food on the counter, the limitOfStudent value does not change. I used 3 different mutex in the student section. Their descriptions are as follows:

1- I use it among graduate students. If there are more than one graduate student, there may be problems during meals (like buying the same plate). So I do the locking process before the students get the food

2- I do the same for Undergraduate students. It is safer to use 2 separate mutex as it is the priority of graduate students.

3- While the students are buying plates, I do another locking process without class separation.

4- As I explained in the cook section, I use a mutex that locks the bench for the cooks to work safely when sending plates or when the students buy plates.

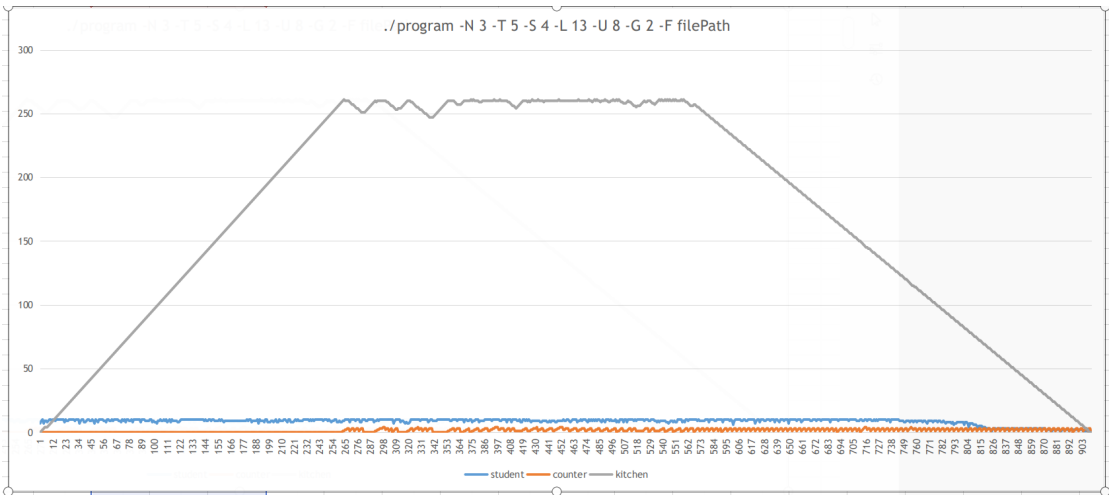
After all these processes, I delete all semaphores running in my program with the help of sem_destroy. And I am freeing the pointer array where I hold the process IDs. Also, if the user makes ctrl + C (SIGINT), the program ends safely. I solved this problem by making a signal handler.

2) GRAPHICAL PLOTS

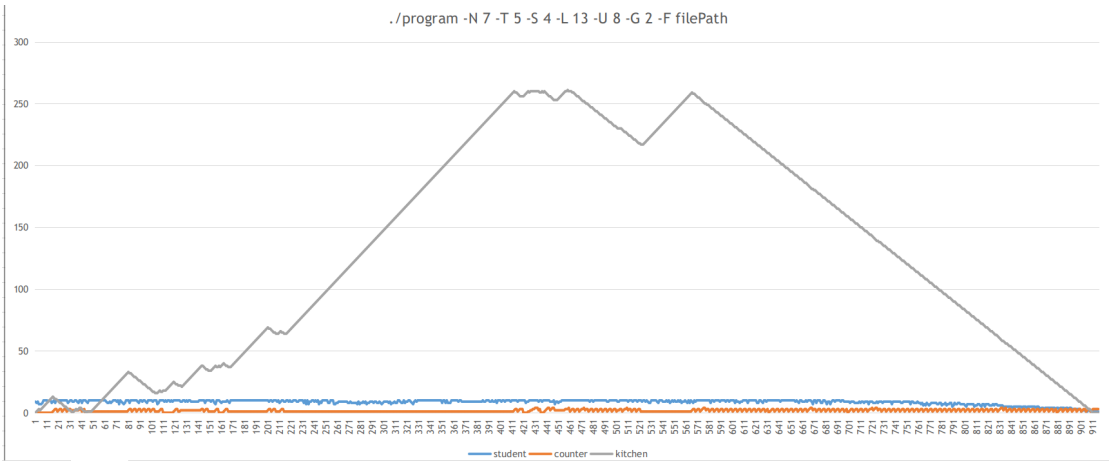
You can see my graphics in graphics folder detailed.

Student => # of students waiting at the counter
Counter => # of plates at the counter
Kitchen => # of plates at the kitchen

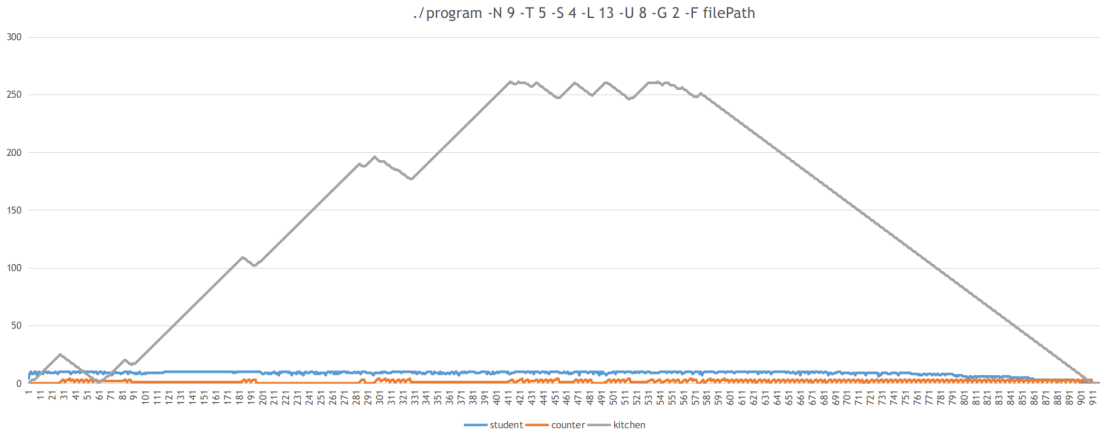
A) Changing Value of N



(a.1)

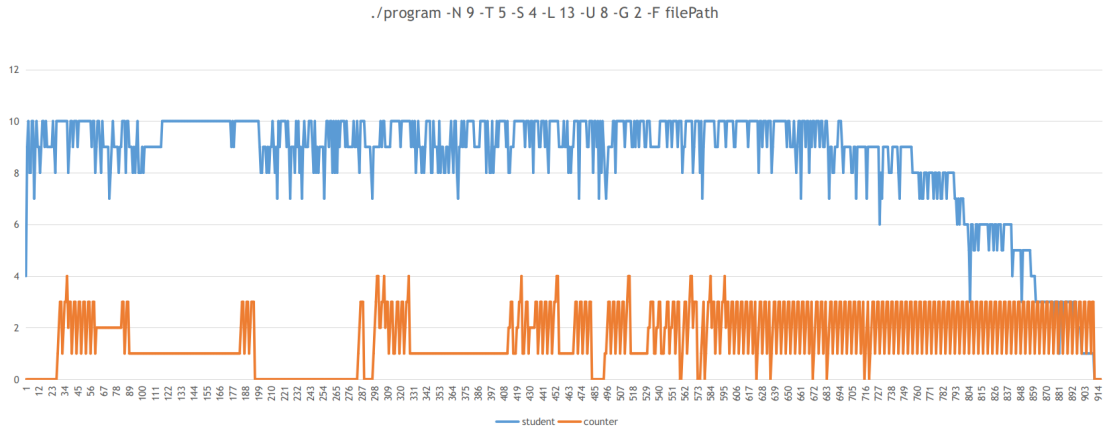


(a.2)



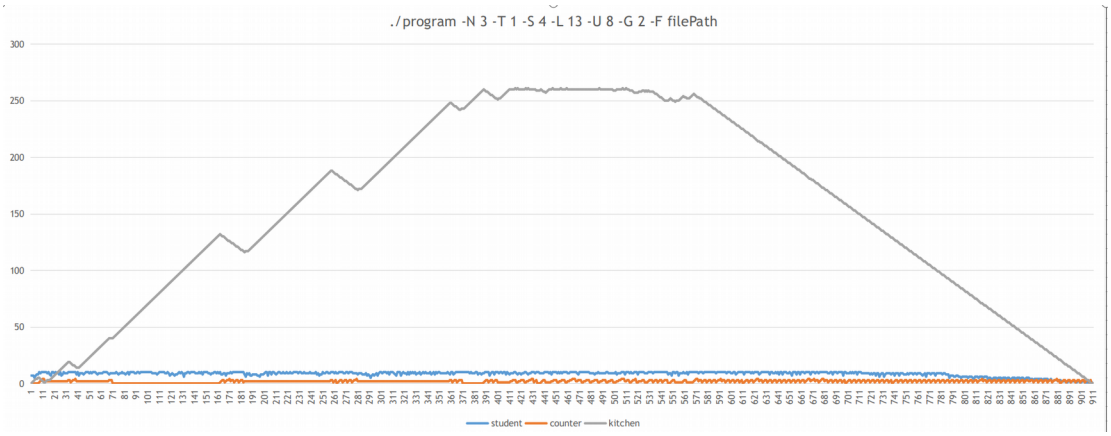
(a.3)

The detailed chart for a.3 is as follows. I just showed the values of # of plates at the counter and # of students waiting at the counter to look detailed.



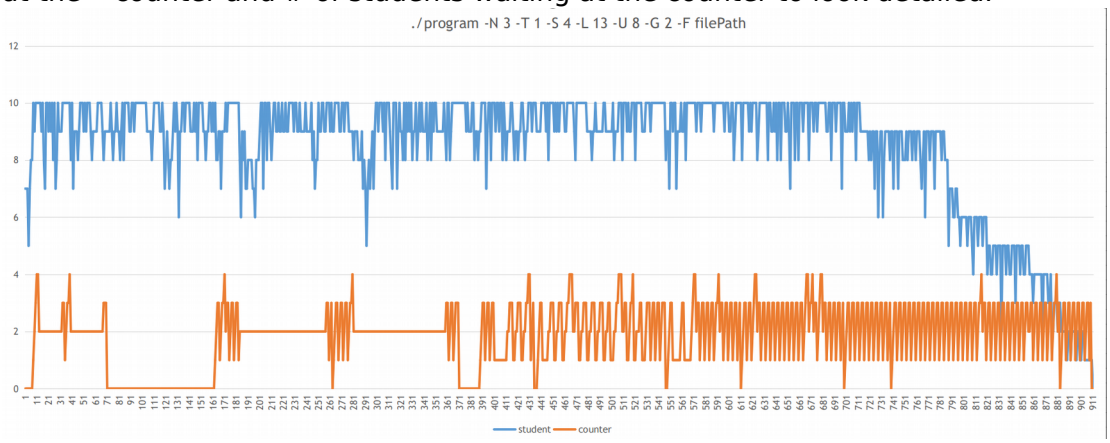
(a.4)

B) Changing Value of T

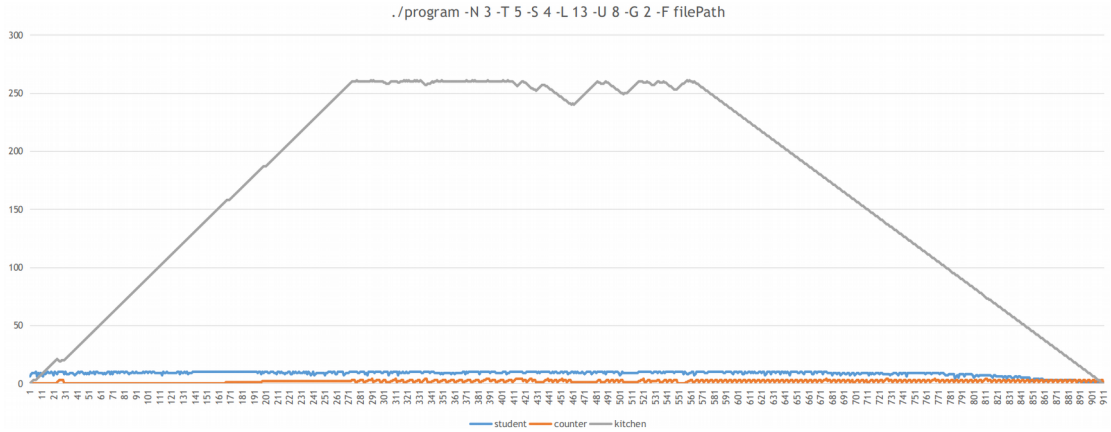


(b.1)

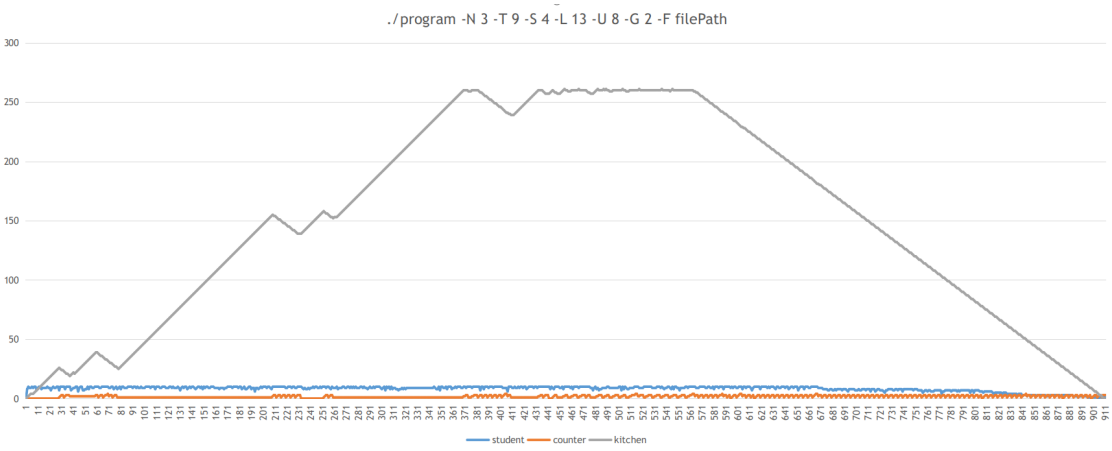
The detailed chart for b.1 is as follows. I just showed the values of # of plates at the counter and # of students waiting at the counter to look detailed.



(b.2)

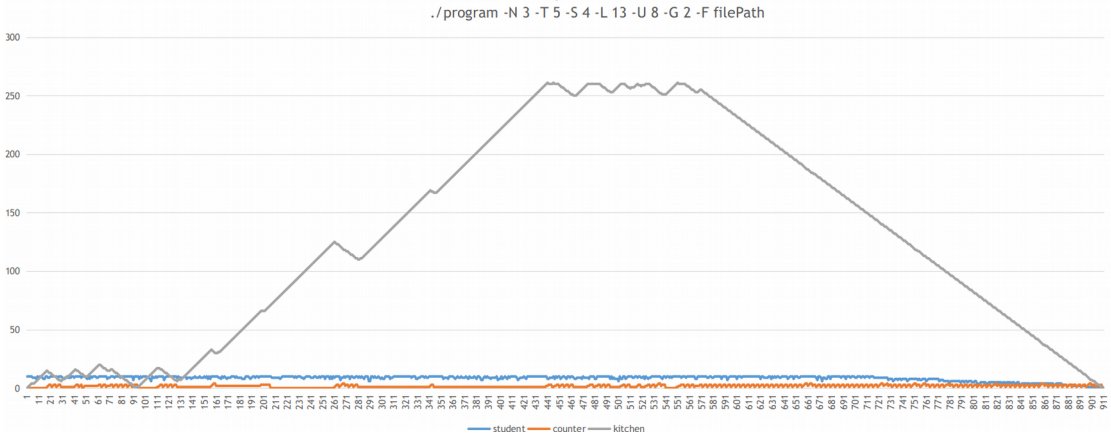


(b.3)

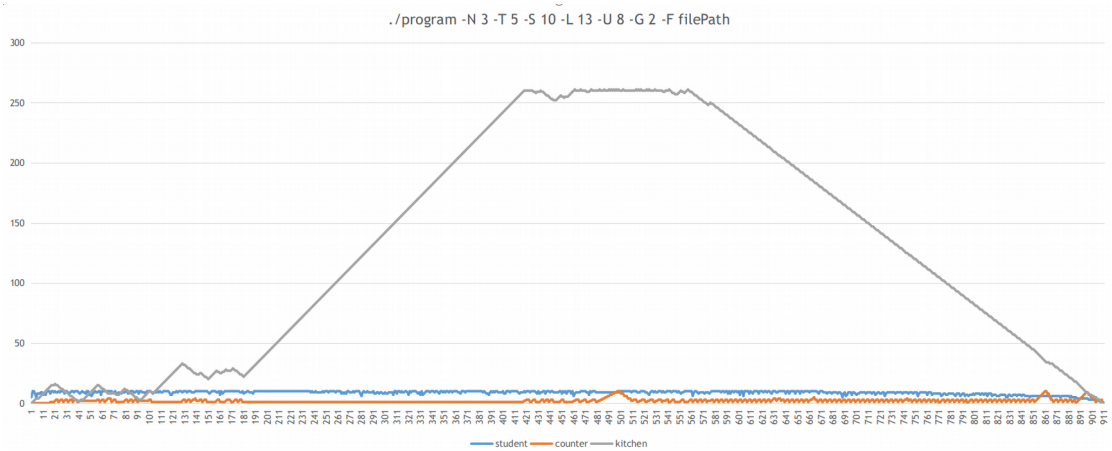


(b.4)

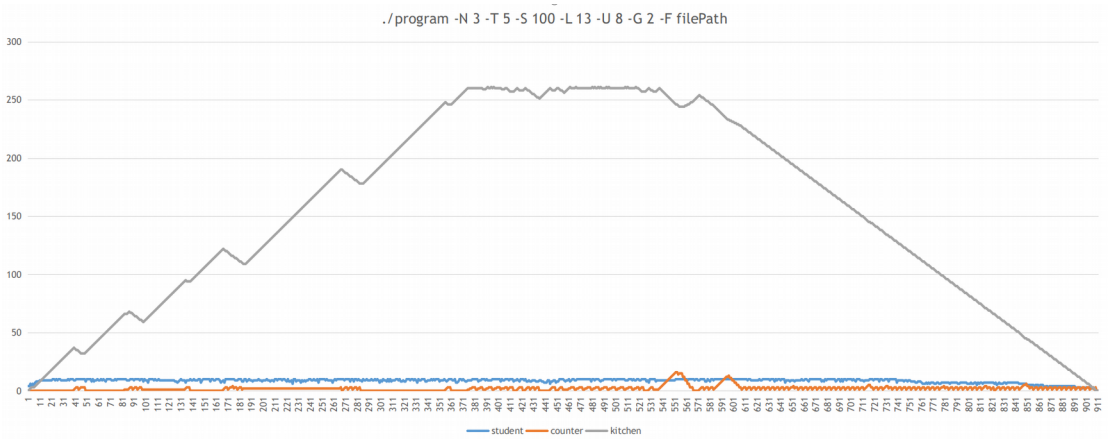
C) Changing Value of S



(c.1)

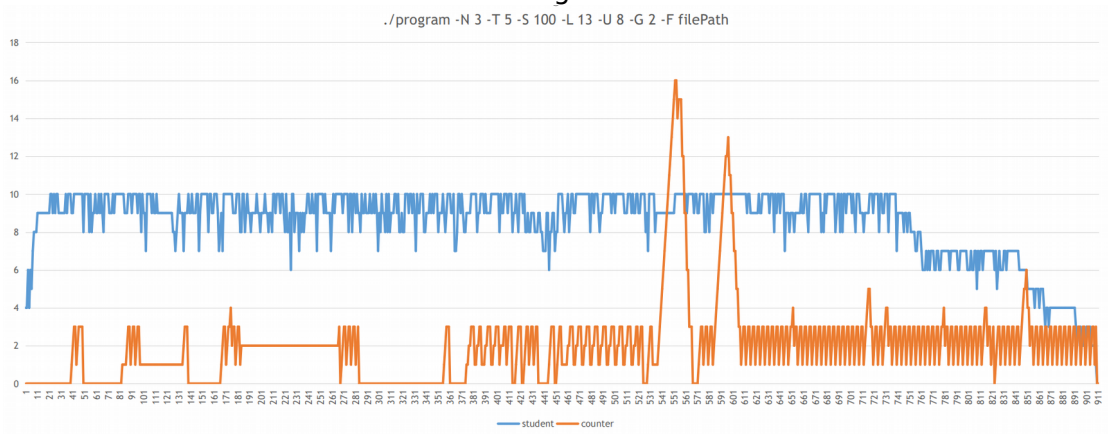


(c.2)



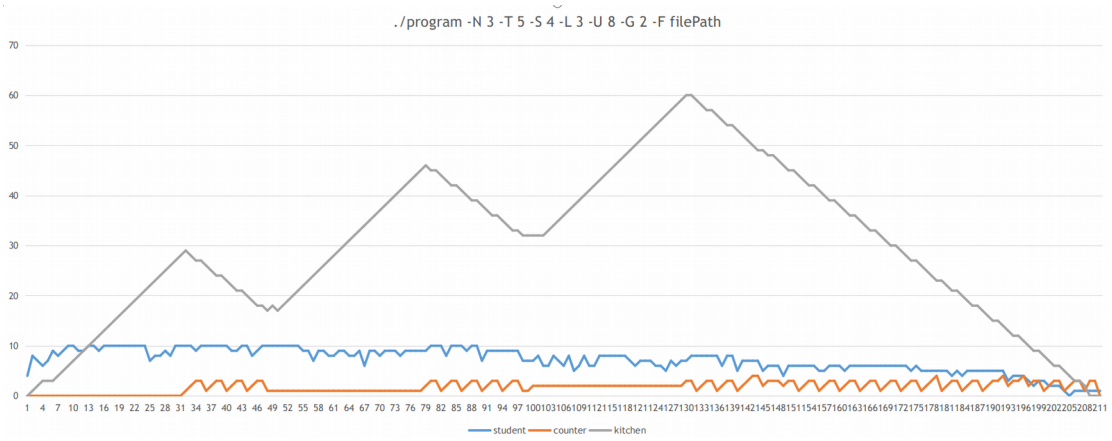
(c.3)

The detailed chart for c.3 is as follows. I just showed the values of # of plates at the counter and # of students waiting at the counter to look detailed.

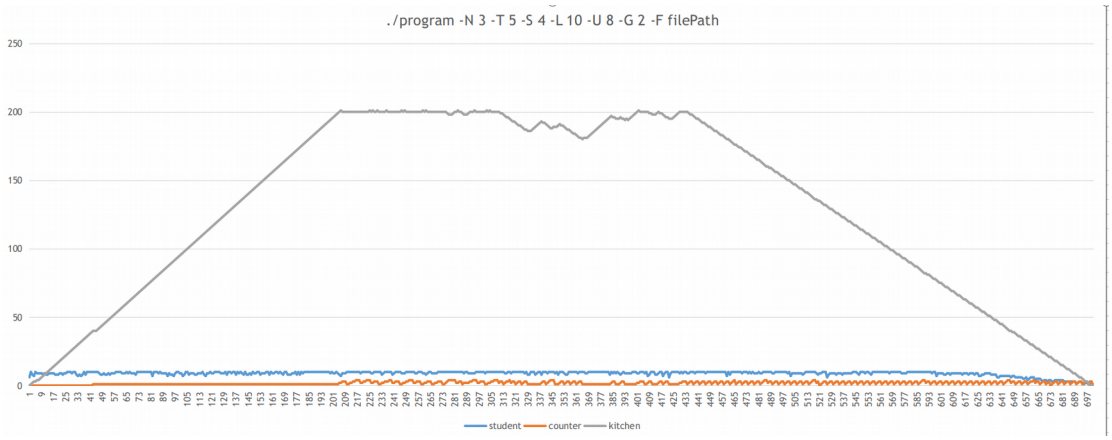


(c.4)

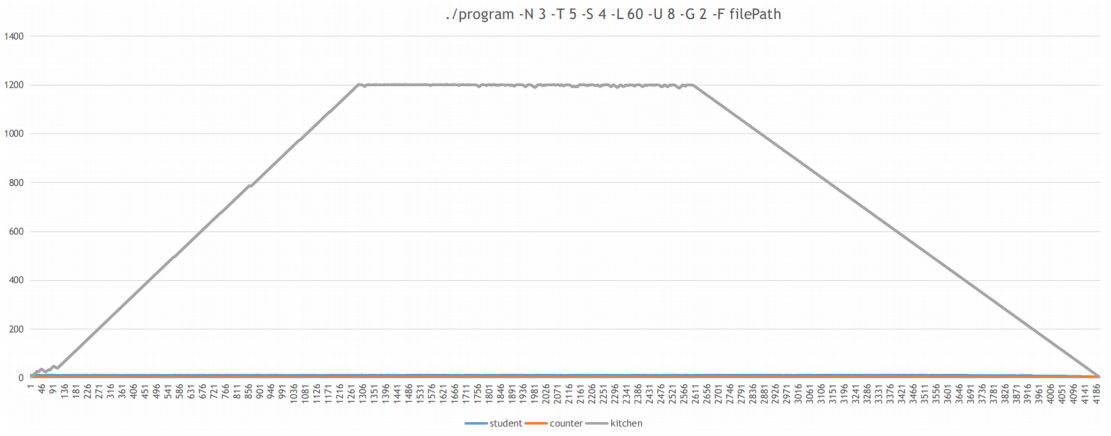
D) Changing Value of L



(d.1)

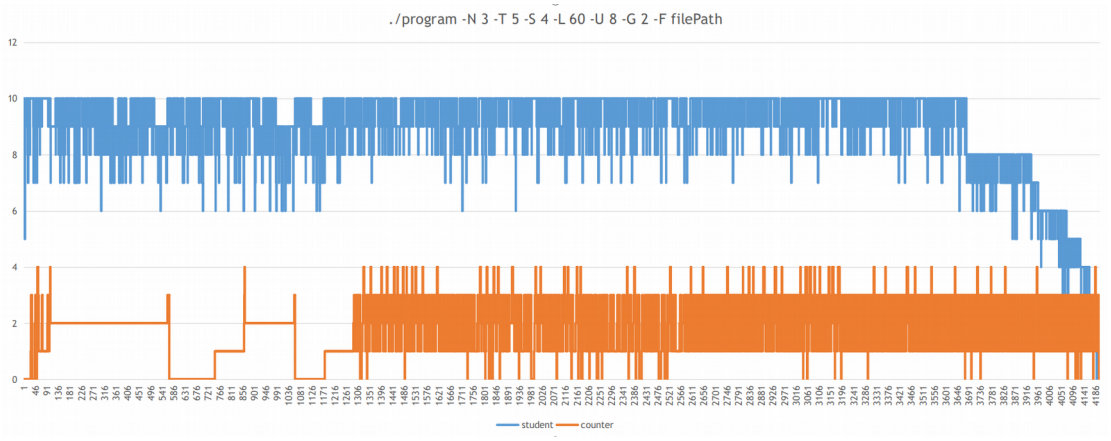


(d.2)



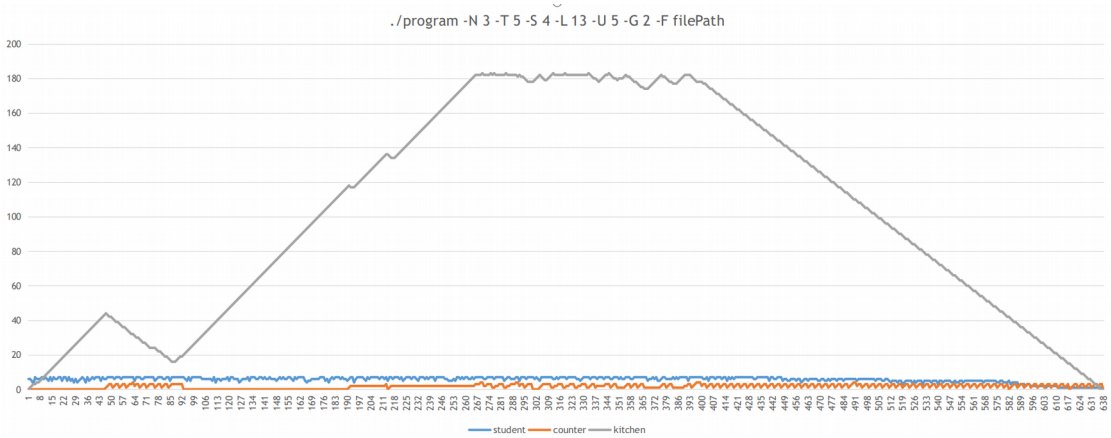
(d.3)

The detailed chart for d.3 is as follows. I just showed the values of # of plates at the counter and # of students waiting at the counter to look detailed.

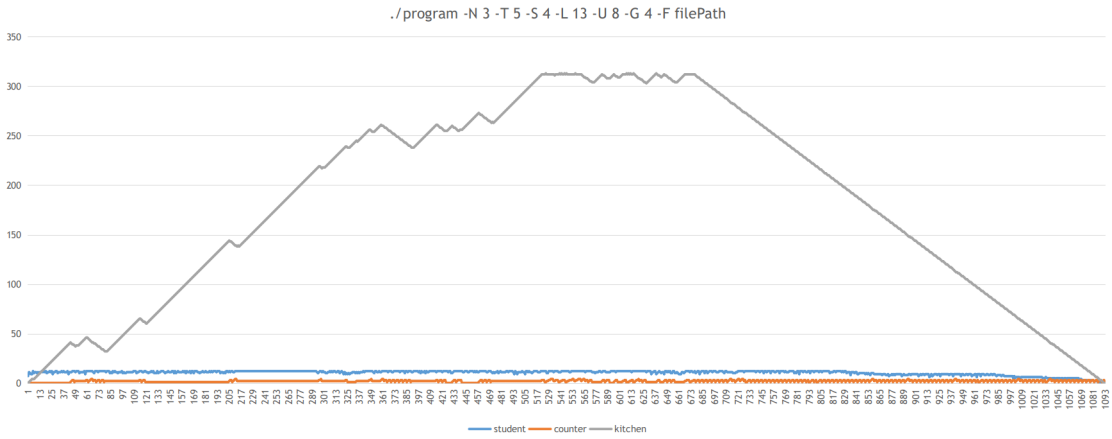


(d.4)

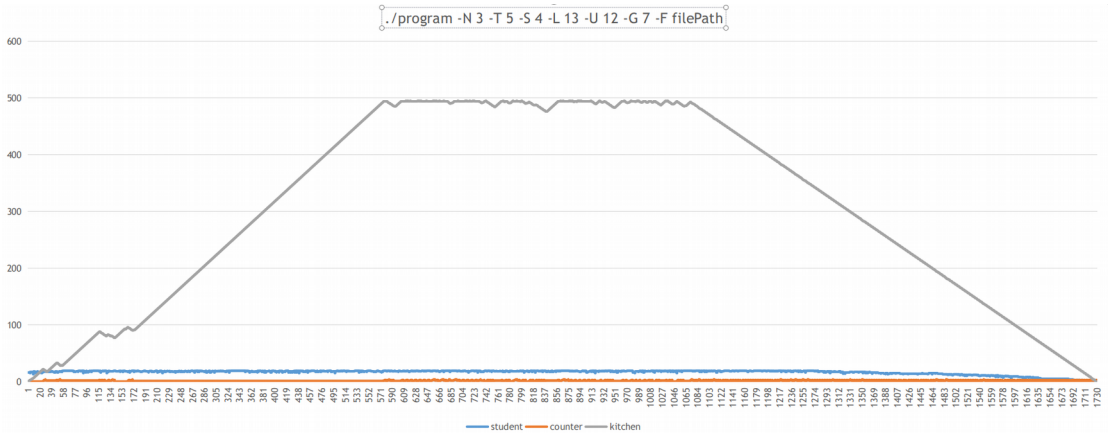
E) Changing Value of U and G (M Parameter)



(e.1)

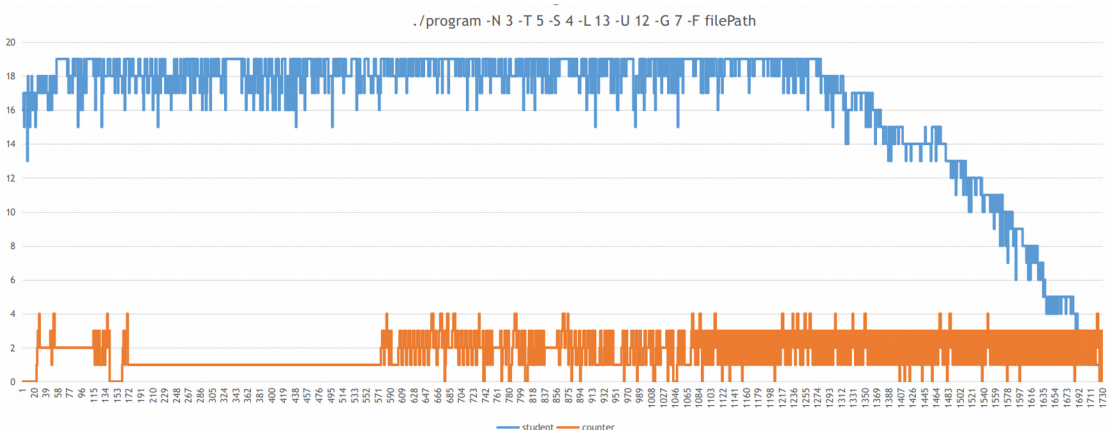


(e.2)



(e.3)

The detailed chart for e.3 is as follows. I just showed the values of # of plates at the counter and # of students waiting at the counter to look detailed.



(e.4)