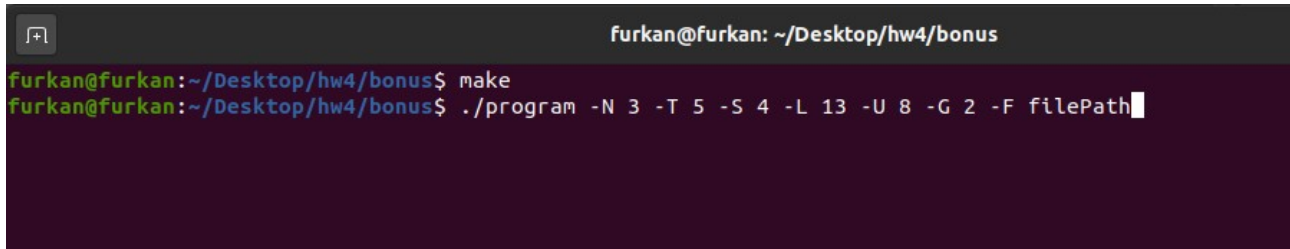


CSE344 – System Programming

REPORT – Midterm Project

Furkan ÖZEV
161044036

COMPILE & RUN: (BONUS)

A terminal window with a dark background. The title bar shows 'furkan@furkan: ~/Desktop/hw4/bonus'. The terminal contains two lines of text: 'furkan@furkan:~/Desktop/hw4/bonus\$ make' and 'furkan@furkan:~/Desktop/hw4/bonus\$./program -N 3 -T 5 -S 4 -L 13 -U 8 -G 2 -F filePath'.

```
furkan@furkan:~/Desktop/hw4/bonus$ make
furkan@furkan:~/Desktop/hw4/bonus$ ./program -N 3 -T 5 -S 4 -L 13 -U 8 -G 2 -F filePath
```

SYNCHRONIZATION PROBLEMS & SOLUTIONS:

PROBLEM 1:

Timer interrupt could come up while printing any output to the terminal. Switching to other processes before the writing process was finished could cause a shift in the output.

SOLUTION 1:

I put the print process in the critical region for all processes. I control these critical regions with mutex called "commonMutex".

Before printing starts, wait (commonMutex) is done, after printing is finished, post (commonMutex) is done, thus taken under control.

PROBLEM 2:

When an error was encountered in any process, I had to print an error message and kill all the processes immediately in order not to cause any deadlock or zombie process problems.

SOLUTION 2:

By using wait (commonMutex) I was preventing all processes from moving after a certain point. Then I print the error message to STDERR.

Using the killpg (0, SIGINT) function, I send the SIGINT signal to the process group, ie all processes.

Thus, when an error occurs, all processes stop, the error message is printed and all processes are terminated.

PROBLEM 3:

Although the kitchen reached its maximum capacity, the supplier was trying to bring products.

SOLUTION 3:

Since the supplier is guaranteed to bring plate to the kitchen, It reserve a place from kitchen for 1 plate. It does this first, using wait(empty). This empty semaphores is used to relation between supplier and cooks.

PROBLEM 4:

Let's consider a scenario where no products are left in the kitchen, but the supplier can bring products. In this case, the cooks could assume that there would be no more products and could end.

SOLUTION 4:

Cooks should analyze this situation and wait for the supplier to bring the product. I used a shared variable. Each time the supplier leaves the product in the kitchen, this variable increases by 1. Cooks check if this number reaches 3LM. If it reaches 3LM, the cooks can finish their job.

PROBLEM 5:

Despite the small number of places on the counter, more than the required number of cooks could go to the kitchen to bring plate and this situation cause deadlocks.

SOLUTION 5:

Since the cook is guaranteed to bring plate from kitchen to the counter, It reserve a place from counter for 1 plate. It does this first, using wait(empty). This empty semaphores is used to relation between cooks and students.

PROBLEM 6:

Despite the small plates amount on the kitchen, more than the required number of cooks could go to the kitchen to bring plate. Some will be able to bring the plates, but some will wait because there are no plates. This caused the cooks to wait forever because all the plates were exhausted. So, this situation cause deadlocks.

SOLUTION 6:

I used a shared variable called "totaltake" for this. If the cook can bring 1 plate from the kitchen, this increased by 1. The cook looks at this variable before going to the kitchen. If this variable has reached 3LM, it will finish its job because there are no plates to bring. Thus, it does not wait unnecessarily and does not cause deadlocks.

PROBLEM 7:

Even when the counter was full, there were no 3 types of plate. Therefore, the students could not get plates, and the cooks could not bring plate because the counter was full. So, this situation cause deadlocks.

SOLUTION 7:

When any cook went to the kitchen before, I guaranteed that cook would bring a plate using semaphore. The cooks decide which type plate to bring before going to the kitchen. There is a simple tour logic when determining the plate to be taken from the kitchen. The same type of plate is brought from the kitchen every 3 times. Thus, even if the counter is full, at least 3 varieties will be found. After the cook decides which plate to wait, cook waits the plate using the semaphore of that plate with wait (). These plate semaphores are increased when placed in the kitchen by the supplier.

PROBLEM 8:

A student will wait until at least one soup, one main course and one desert are available on the counter, in which case she/he will take all 3 of them at once and leave the counter. So student will either take all three plates together (P, C and D) or none.

SOLUTION 8:

I created a semaphore called "meal" that would allow synchronization between students and cooks. If 3 types of plates are supplied to counter, meal semaphore is increased by 1. Thus, the student can consume the meal with using this semaphore. There are 3 shared temporary variables to set this synchronization. When the cook adds plates to the counter, this temporary variable of the plate also increases by 1. Each time a plate is added to the counter, it is checked whether the values of these variables are all greater than 0. If the condition is met, that is, if there is at least 1 of each variety, the meal semaphore is increased by post, and these temp variables are reduced by 1.

PROBLEM 9:

The student will find an empty table, sit down and eat, if there are no empty tables, the student will wait for one. After eating, the student will once again go to the counter and repeat this behavior L times.

SOLUTION 9:

I have separate semaphores called "empty", "full" and "mutex" that will allow students to sit at the table. In this problem, we can think student is consumer as the student sits at the table and as the producer when student leaves the table. There is a certain amount of tables and these will be empty in the first stage. For this, I created a shared array in the number of tables and assigned 0 to all its elements. I used 0 to represent that the table is empty, and 1 to represent that the table is full. The student waits until there is an empty table, then finds which table is empty by navigating the

array. and makes it 1. When student left the table, student sets the value in this index to 0. Then student moves on to the new round.

PROBLEM 10:

The student will find an empty table, sit down and eat, if there are no empty tables, the student will wait for one. After eating, the student will once again go to the counter and repeat this behavior L times.

SOLUTION 10:

I have separate semaphores called "empty", "full" and "mutex" that will allow students to sit at the table. In this problem, we can think student is consumer as the student sits at the table and as the producer when student leaves the table. There is a certain amount of tables and these will be empty in the first stage. For this, I created a shared array in the number of tables and assigned 0 to all its elements. I used 0 to represent that the table is empty, and 1 to represent that the table is full. The student waits until there is an empty table, then finds which table is empty by navigating the array. and makes it 1. When student left the table, student sets the value in this index to 0. Then student moves on to the new round.

PROBLEM 11 – (BONUS PROBLEM):

The students wait in front of the counter. There is no queue or line of any form at the counter. Graduate students have priority at the counter over undergraduates. As long as there are graduates in front of the counter waiting for their food, no undergraduate is allowed to be serviced.

SOLUTION 11:

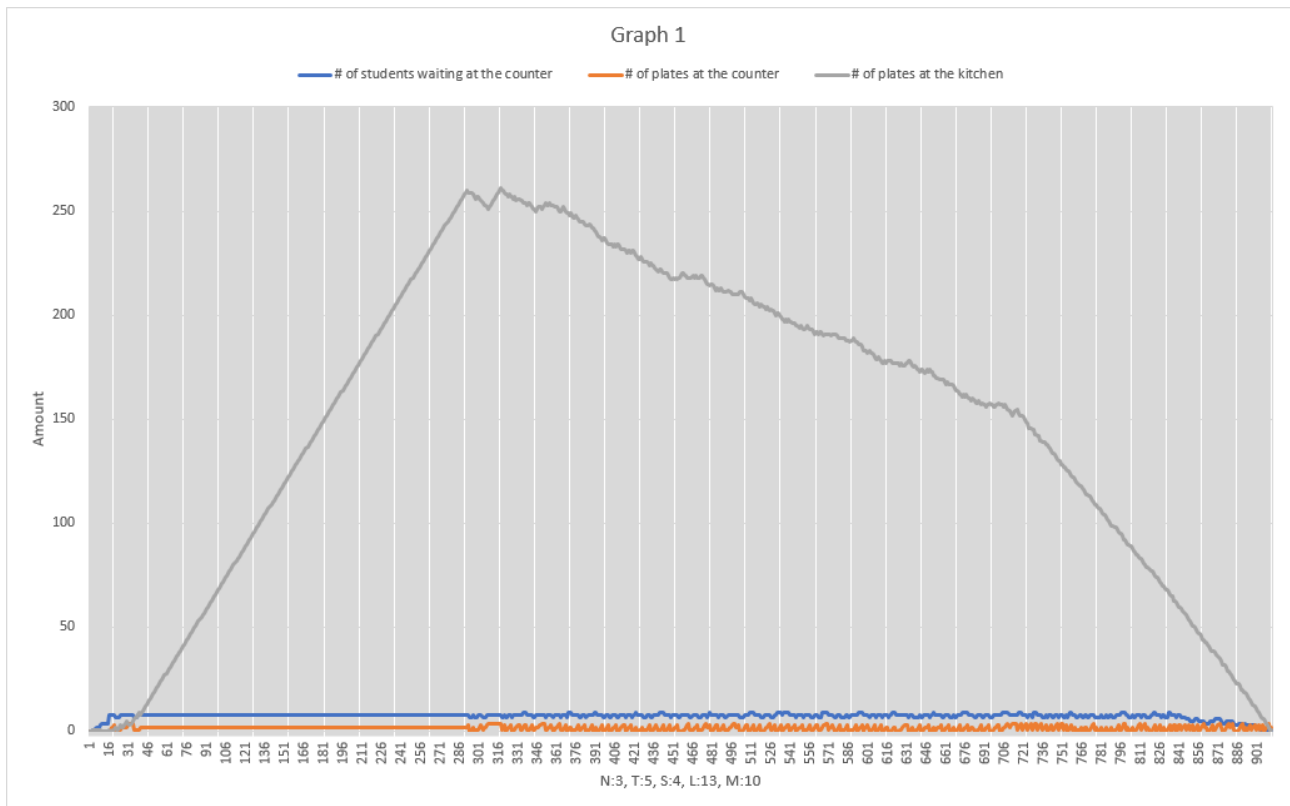
I used "UGmutex" for the relationship between undergraduate and graduate students. I used "countG" is used for keeping number of graduate students that waiting at counter.

I solved this problem simply as follows: "UGmutex" is locked when there is a graduate student at the counter, and it is unlock when there is no graduate student. Undergraduate student waits for this lock to be unlocked in order to service.

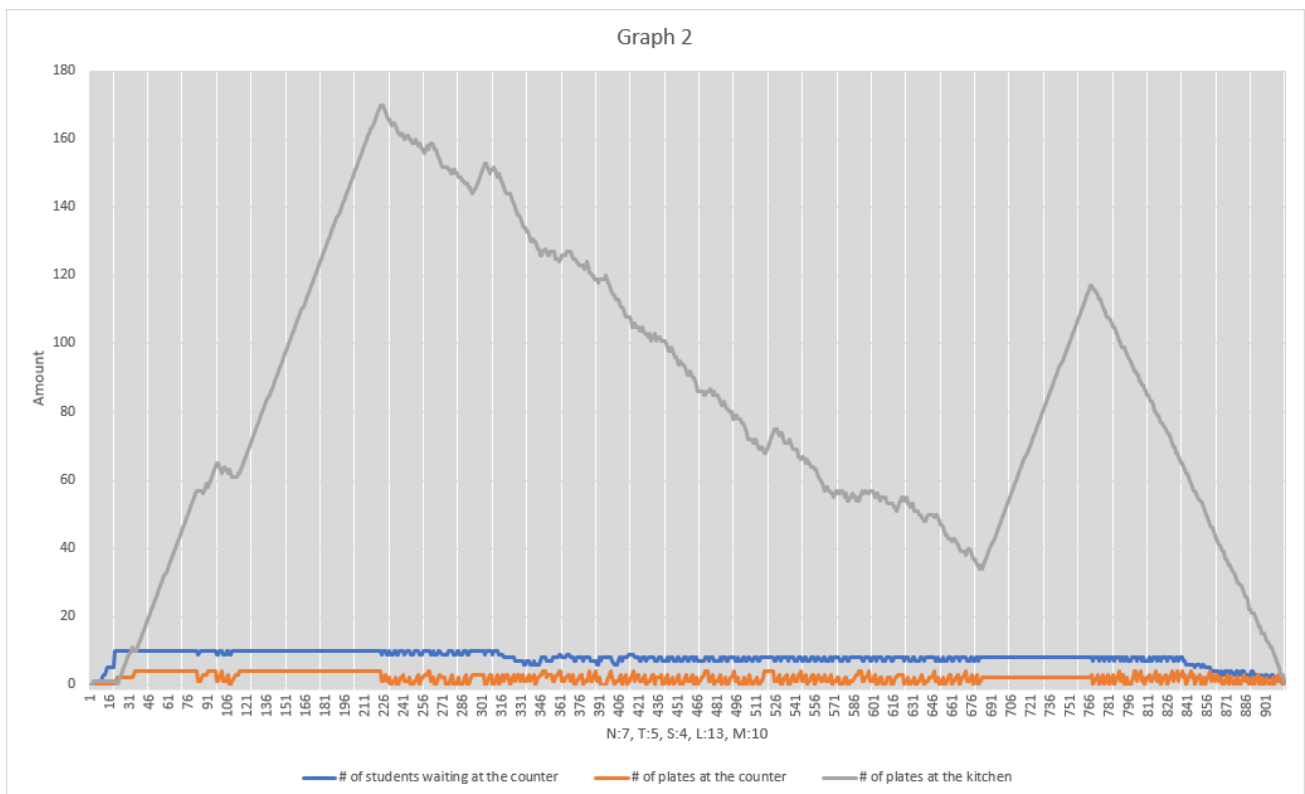
If there are no graduate students at the counter, the first-come graduate student locks this mutex. The graduate student who left the counter lastly, unlocks this mutex. If there are no graduate students, the undergraduate student locks this mutex and turns unlock after receiving the meal.

GRAPHICAL PLOTS:

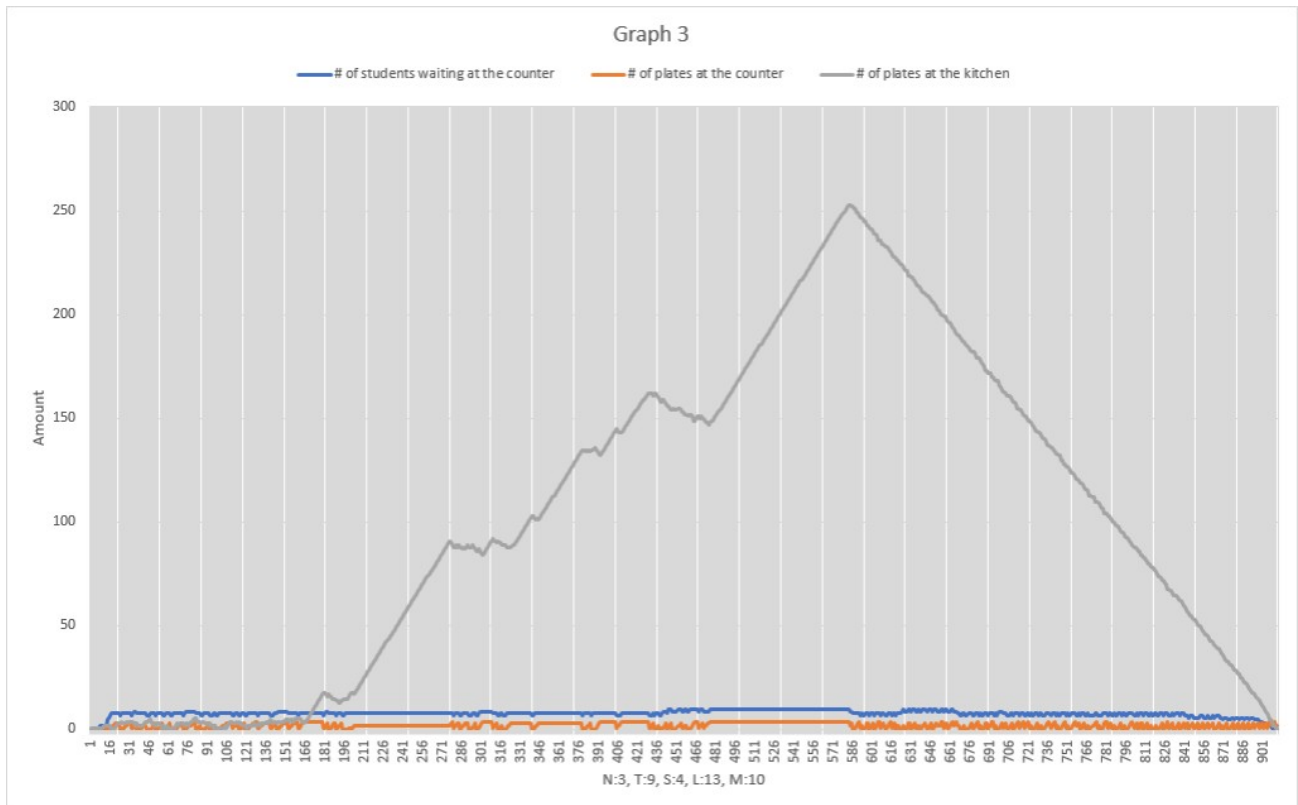
GRAPH 1:



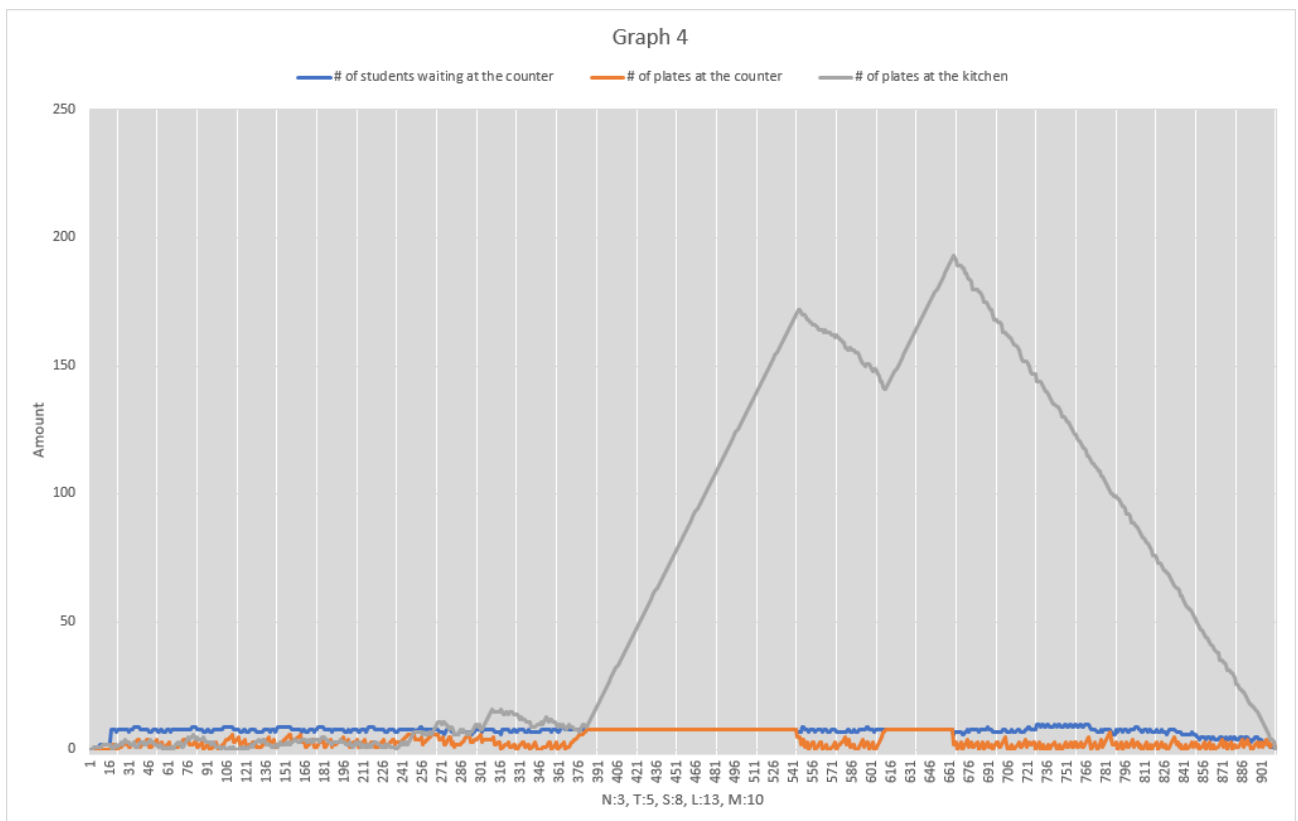
GRAPH 2:



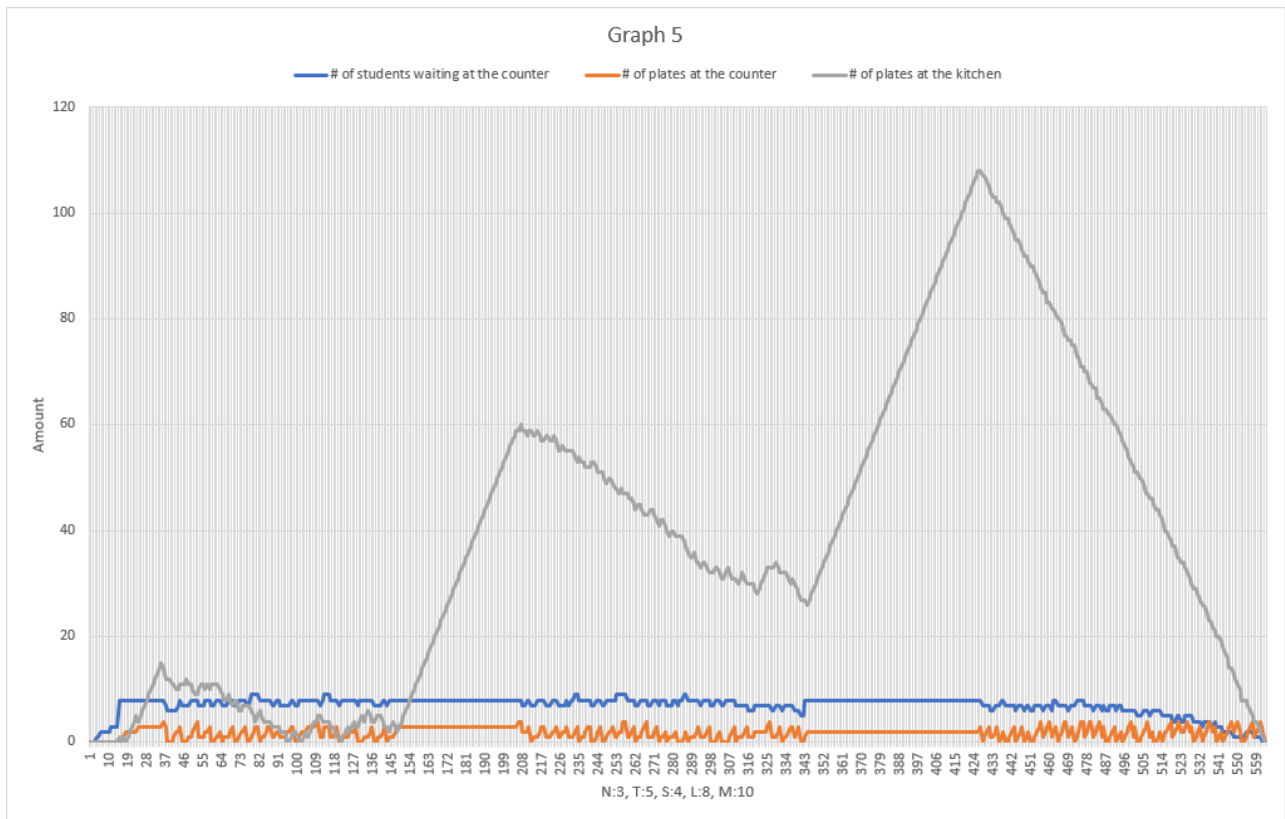
GRAPH 3:



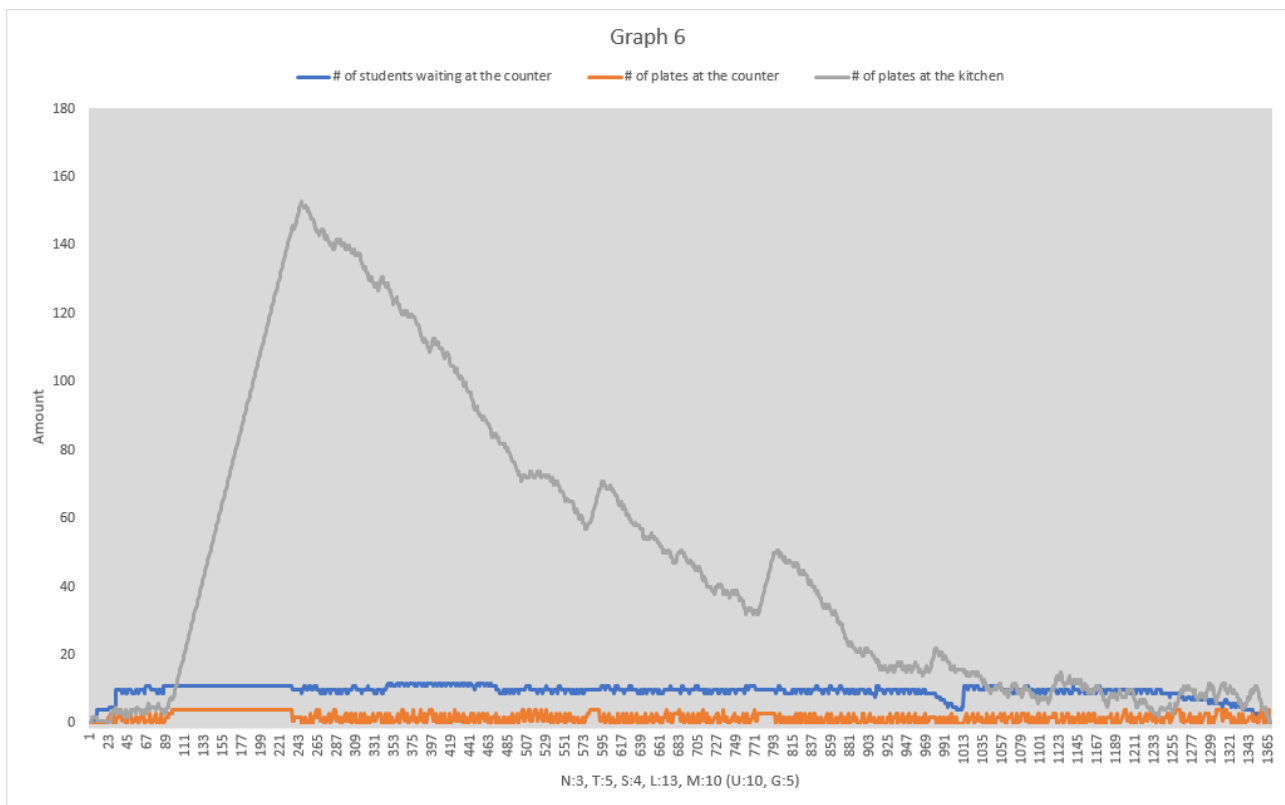
GRAPH 4:



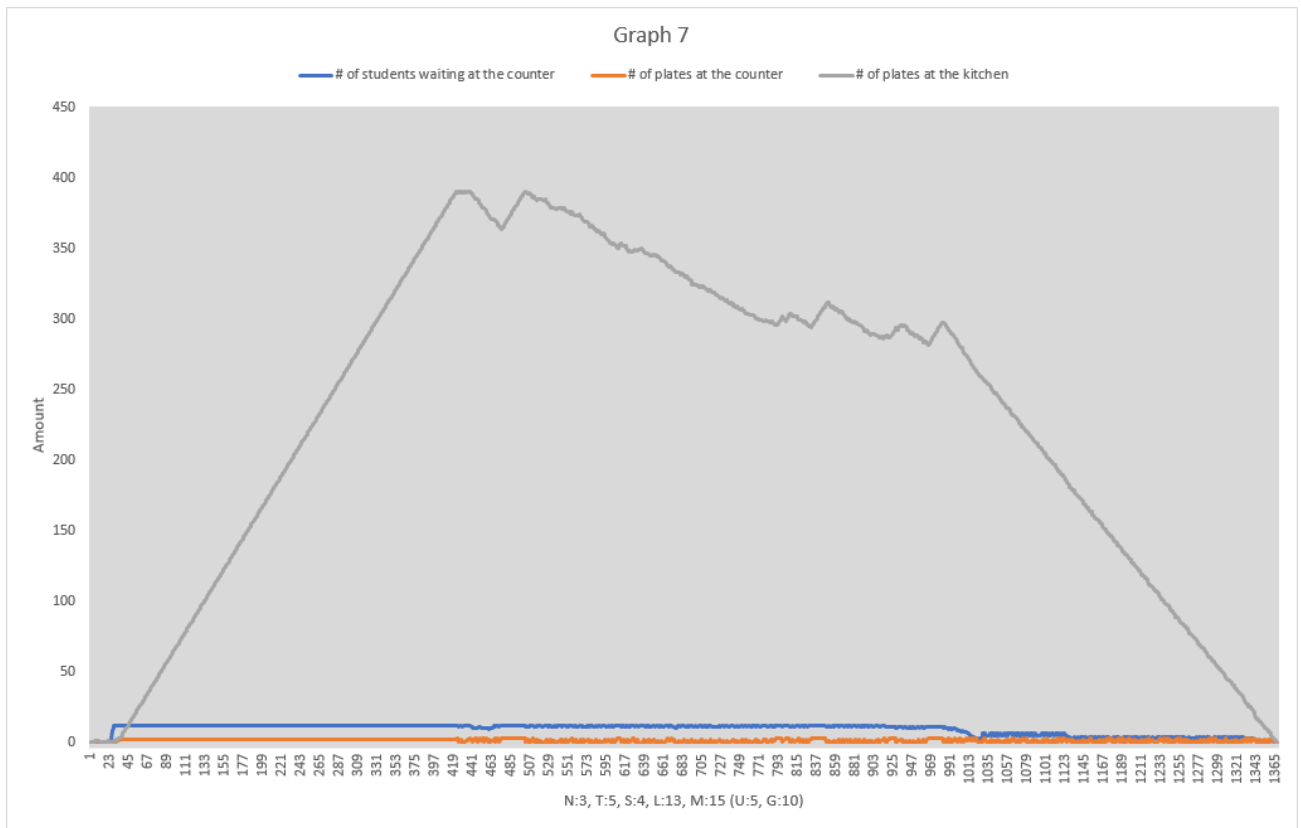
GRAPH 5:



GRAPH 6:



GRAPH 7:



GRAPH 8:

