# Gebze Technical University Computer Engineering

# CSE 344
# System Programing

# MIDTERM PROJECT REPORT

ELİF AKGÜN
1801042251

# 1. Problem Definition

In this project, I simulated the student mess hall of a university. There are 3 actors: the supplier, the cooks and the students. There are 3 locations: the kitchen, the counter and the tables.

Example command line argumenst for this project:

```
$./program -N 3 -M 12 -T 5 -S 4 -L 13 -F filePath
```

There are N cooks, M students, T tables, a counter of size S, and a kitchen of size K. Supplier delivers plates to kitchen and cooks take plates form there. Each plates can be 3 types: P(soup), C(main course), and D(desert). Each student eats these foods L times. filePath denotes the file that includes P, C and D types plates 3*L*M times. There are some constraints:

S > 3
M > T >= 1
L >= 3
K = 2LM+1

## 1. *Supplier*

Supplier deliveres exactly 3*L*M plates to kitchen (L*M plates P, L*M plates C, L*M plates D) in total and at each visit to the kitchen supplier delivers only one plate. If there is no room left at the kitchen, supplier waits for room to open up. The supplier reads file and each char that read represents food types. So supplier delivers plates to kitchen in an arbitrary order and terminates when it reaches end of the file.

The supplier prints 3 types of messages:

```
// Entering the kitchen
The supplier is going to the kitchen to deliver soup: kitchen items
P:3,C:4,D:3=10

// Afer delivery
The supplier delivered soup – after delivery: kitchen items P:4,C:4,D:3=11

// Done delivering.
The supplier finished supplying – GOODBYE!
```

## 2. *Cook*

Each cook, if there is one, gets each plates from the kitchen then places the plates on the counter if there is any room left. If there is no room, cook wait for room to open up. Each cook repeats this. After finishing placing all plates cooks terminate.

Each cook process will print 4 types of messages:

// waiting for/getting deliveries
Cook 0 is going to the kitchen to wait for/get a plate - kitchen items P:4,C:4,D:3=11
// Going to the counter
Cook 0 is going to the counter to deliver soup – counter items P:2,C:1,D:1=4

// After delivery to counter
Cook 1 placed desert on the counter - counter items P:2,C:1,D:2=5

// After finishing placing all plates
Cook 0 finished serving - items at kitchen: 0 – going home – GOODBYE!!!

### *3. Student*

Students wait counter until at leaat one soup, one main course an done desert are available on the counter then take all 3 of them at one and leave. If there are one type of food is available, students don't take this and wait. There is no queue at the counter. Then the student will find an empty tablei sit down adn eat. If there are no empty tables, the student waits for one. After eating, students repeat this again. In total, each student eats L times.

Each student process will print 5 types of messages: (round x) where 1 <= x <= L

// arriving at the counter/waiting for food
Student 1 is going to the counter (round 1) - # of students at counter: 1 and counter items P:2,C:1,D:2=5

// waiting for/getting a table
Student 0 got food and is going to get a table (round 1) - # of empty tables: 4

// sitting to eat
Student 0 sat at table 1 to eat (round 4) - empty tables:1

// done eating, going again to the counter; times x, where x is increased to x+1
Student 0 left table 1 to eat again (round 2) - empty tables:2

// after finishing eating L times
Student 2 is done eating L=5 times - going home – GOODBYE!!!

## 2. Implementation

I implement every actor as a separate process. In kitchen part, so in the supplier/cooks relationship I benefited from producer consumer problem and in counter part, so in cooks/students relationship I benefited from cigarette smokers problem. Main process initializes semaphores, memory allocation for shared memory then forks the child processes then waits for them.

### 1. *Supplier*

In supplier process, first of all, supplier opens file denoted by filePath and reads file. Each file represents the 3 type of plates: P, C or D. Supplier delivers plate that read form file as a char. I encountered a synchronization problem in this process. This problem is that there is no room left in the kitchen. I used producer/consumer problem solution for this problem. First, supplier checks the kitchen wheter there is any room left in the kitchen. For this operation I used a semaphore denoted by *rooms_kitchen*. If there is any room at the kitchen, supplier takes lock denoted by *kitchen_lock* ( *kitchen_lock* is a binary semaphore) to block cook then delivers. For wait and block operations, I used semaphores. After each delivery, supplier post to *kitchen_lock* and *items_kitchen*. Supplier repeats these operations 3*L*M times in a for loop and prints messages on the screen and after delivery it exits.

### 2. *Cooks*

In cook processes, I encountered 3 synchronization problems. First of these problems is that there is no item in the kitchen. . I used producer/consumer problem solution for this problem. First of all, cook waits for supplier if there is no item in the kitchen. For this purpose I used *items_kitchen* semaphore. If there is at least one suitable food in the kitchen, cook takes lock denoted by *kitchen_lock* to block supplier, take this plate and delivers to counter. After each delivery counter post to *kitchen_lock* and *rooms_kitchen.*

Cook must check the plates on the counter. If the counter is full, containing exactly S items, then at least 3 of them will be distinct (one soup, one main course and one desert), and thus at least one student will be able to take them and eat. This is the second problem. I encountered this problem with *P_kitchen*, *D_kitchen* and *C_kitchen* semahores and *counter_P_count*, *counter_C_count* and *counter_D_count* variables. This variables indicate number of each type of plates at the counter. Each time the cook goes to the kitchen waits for suitable food, whichever plate is less on the counter cook takes that plate. So counter contains exactly S items at least 3 of them, when it is full.

I used a loop in each cook process. So cooks repeat these operations until there is no element left in the kitchen. When the cooks looked at the plates on the counter at the same time and went to buy them from the kitchen at the same time, problems appeared in the plate types. For example, Cook 0 knows that P:1, C:1 and D:1 and goes to kitchen to take D plate, then Cook 1 arrives to counter and it also delivers D plate so after delivery of Cook 1, P:1 C:1 D:2 at the counter. Then a student take this three plates, P:0, C:0, D:1 at the counter now. Cook 0 delivers D plate again, bu there is also a D plate at the counter. It should delivers P plate or C plate to a student take. This is the third problem and I used *cook_lock* binary semaphore fort his problem. In this way, there is only one cook in the critical section so each cook takes right plate form the kitchen.

In these process, each cook process prints 4 types of messages and exit.

### 3. *Student*

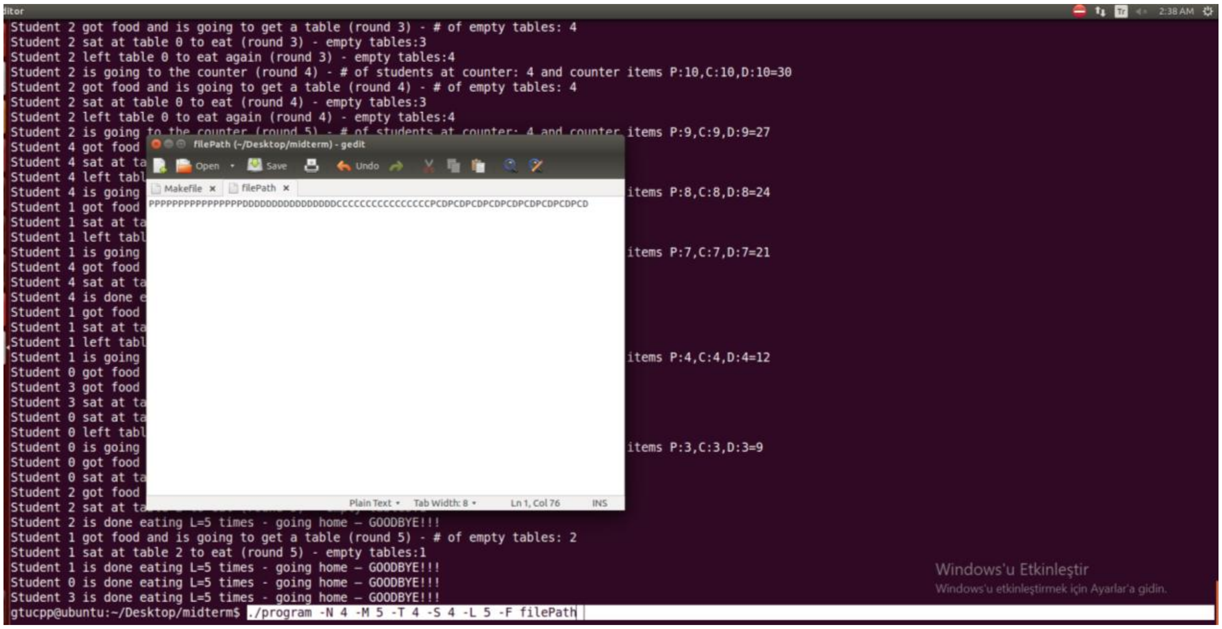
In student processes, I encountered many synchronization problems. To encountered these problems, I benefited from cigarette smokers problem solution for posix semaphores which is described in the lesson. I forked M process for all students, and 6 helper processes: 3 pusher processes and 3 agent processes. In the cigarette somokers problem there are 3 smokers and in the solution, there is an assumption. Assumption is that each smokers have a type of plate infinitely. But this Project this assumption is wrong. No student has a type of plate. For this reason, first of all, each student waits for P plate, C plate or D plate. I controlled this with indexes. If index%3 = 0, student waits for P plate; else if index%3=1,student waits for C plate; else student wait for D plate. Then students wait for *P_counter2, C_counter2 or D_counter2*. These are semaphores and if value of these semaphores more than zero, at least one child can eat. For example, Student 1 waits for *P_counter* and pass this line, then waits *P_counter2.* If it passes this line it means that one of pusher post to *P_counter2* semaphore. Now, Student 1 has each 3 type of plates and can eat. After student takes food, waits for empty table. I controlled that with e*mpty_table* semaphore. I used *table* integer array that's size is T elements are 0 intially. When a student take foods, looks for one empty table with this array. If one of array elements is 1 ,it means that this table is full. After student eats food, does table[index]=0 again, it means that this table is empty now. Then student goes to counter again to eat again and repeat these operations until L times eat. While these proceses, students prints 5 types of messages on the screen and exit when each student eats L times.

Agent processes waits for P plates, C plates and D plates. Agent 1 waits for both P plate and C plate; Agent 2 waits for both P plate and D plate; and Agent 3 waits for both C plate and D plate. Then each agent process post to *P_counter3, C_counter3 or P_counter3.* For example Agent 1 waits for *P_counter* then post *P_counter3* and *C_counter* then post *C_counter3*. In this way agent reports to pushers there is a P/C/D plate on the counter. Agents repeat these operations until there is no P, C, or D plate.

Pusher processes waits for *P_ counter3*, *C_ counter3* or *D_ counter3*. Pusher 1 waits for *P_ counter3,* Pusher 2 waits for *C_ counter, and* Pusher 3 waits for *D_ counter3.* Then each pocess take lock denoted by *pusher_lock (*binary semaphore) to block other pushers. In critical section, pushers checks wheter checks if there are other type of plates on the counter. If there are, post to *P_counter2, C_counter2 or D_counter2;* else indicates that there is a specified type of plate. To indicate these, I used boolean variables: *isSoup*, *isMainCourse* and *isDesert*. For example, firstly, Pusher 1 waits for P_counter3, and Pusher 2 waits for C_counter3. Then Agent 1 post *P_counter3* and *C_counter3*. Let's Pusher 1 runs before and takes *pusher_lock*. It checks wheter there is other type of plates. If *isMainCourse==1,* it post *D_counter2,* else if *isDesert==1,* it post *C_counter2,* else it indicates that there is a soup by isSoup equals 1. Then post to *pusher_lock*. After Pusher 1, Pusher 2 runs and take *pusher_lock.* It checks is there any soup. Yes, there is a soup( isSop=1), then post *D_counter2,* then post the *pusher_lock*. Now, isSoup=0.
In addition, I encountered another synchronization problem. For example, Pusher 1 take P_counter3 and isSoup=1, then another agent post *P_counter3* and Pusher 1 take this and isSoup=1 again. One of P plate goes on. To avoid this I used *lock1, lock2* and *lock3* binary semaphores. In that way, when the same type of plate comes without using existing same tyepe of plate, it will wait. This is how to pushers work. All pushers work until each student eats, then exit.

## 3. Tests

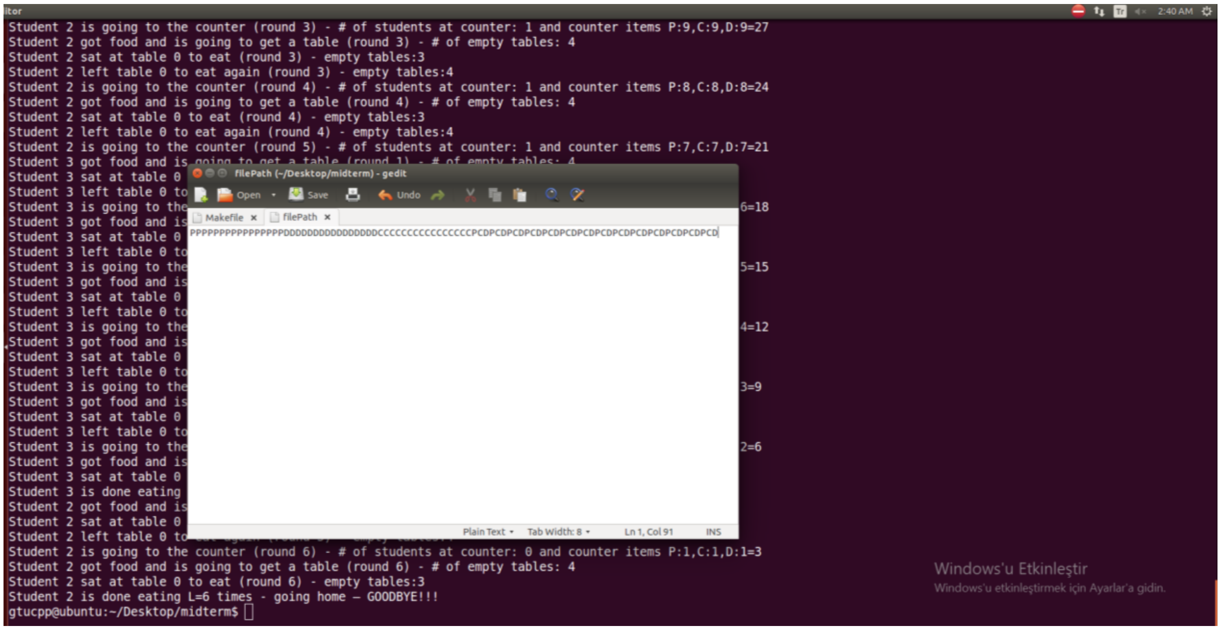
### 1. *Sample Input and Output 1*



```
Student 2 got food and is going to get a table (round 3) - # of empty tables: 4
Student 2 sat at table 0 to eat (round 3) - empty tables:3
Student 2 left table 0 to eat again (round 3) - empty tables:4
Student 2 is going to the counter (round 4) - # of students at counter: 4 and counter items P:10,C:10,D:10=30
Student 2 got food and is going to get a table (round 4) - # of empty tables: 4
Student 2 sat at table 0 to eat (round 4) - empty tables:3
Student 2 left table 0 to eat again (round 4) - empty tables:4
Student 2 is going to the counter (round 5) - # of students at counter: 4 and counter items P:9,C:9,D:9=27
Student 4 got food
Student 4 sat at ta
Student 4 left tabl
Student 4 is going                                                                    items P:8,C:8,D:8=24
Student 1 got food
Student 1 sat at ta
Student 1 left tabl
Student 1 is going                                                                    items P:7,C:7,D:7=21
Student 4 got food
Student 4 sat at ta
Student 4 is done e
Student 1 got food
Student 1 sat at ta
Student 1 left tabl
Student 1 is going                                                                    items P:4,C:4,D:4=12
Student 0 got food
Student 3 got food
Student 3 sat at ta
Student 0 sat at ta
Student 0 left tabl
Student 0 is going                                                                    items P:3,C:3,D:3=9
Student 0 got food
Student 0 sat at ta
Student 2 got food
Student 2 sat at ta
Student 2 is done eating L=5 times - going home – GOODBYE!!!
Student 1 got food and is going to get a table (round 5) - # of empty tables: 2
Student 1 sat at table 2 to eat (round 5) - empty tables:1
Student 1 is done eating L=5 times - going home – GOODBYE!!!
Student 0 is done eating L=5 times - going home – GOODBYE!!!
Student 3 is done eating L=5 times - going home – GOODBYE!!!
gtucpp@ubuntu:~/Desktop/midterm$ ./program -N 4 -M 5 -T 4 -S 4 -L 5 -F filePath
```

```
PPPPPPPPPPPPPPPDDDDDDDDDDDDDDDCCCCCCCCCCCCCCCPCDPCDPCDPCDPCDPCDPCDPCDPCDPCD
```

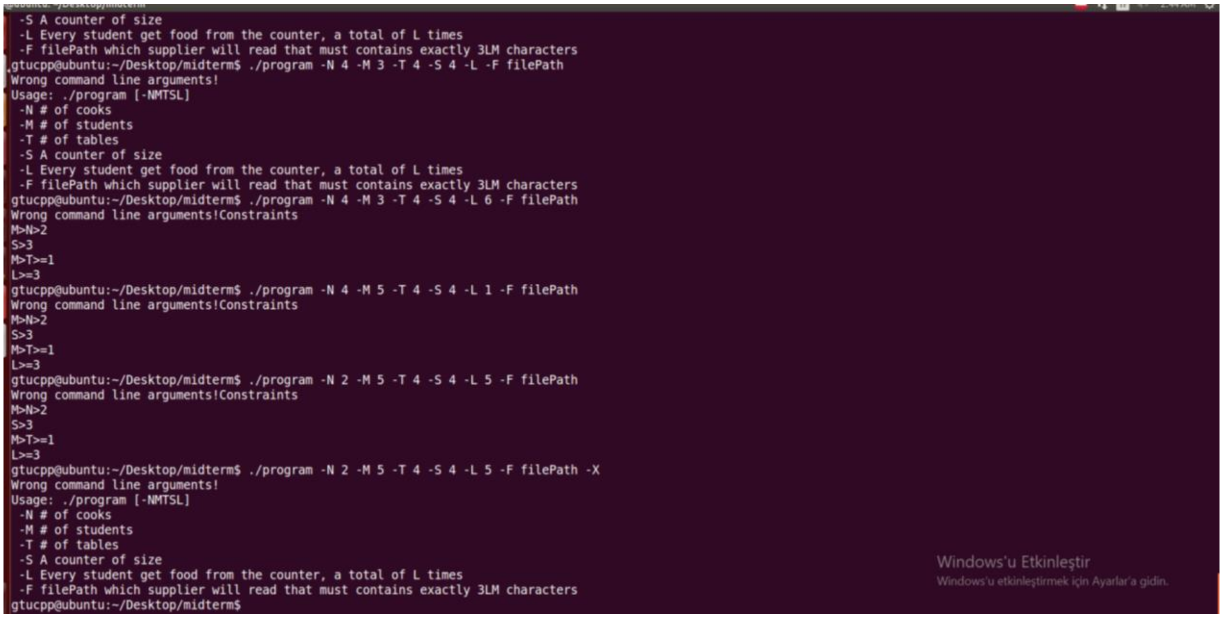
### 2. *Sample Input and Output 2*



```
Student 2 is going to the counter (round 3) - # of students at counter: 1 and counter items P:9,C:9,D:9=27
Student 2 got food and is going to get a table (round 3) - # of empty tables: 4
Student 2 sat at table 0 to eat (round 3) - empty tables:3
Student 2 left table 0 to eat again (round 3) - empty tables:4
Student 2 is going to the counter (round 4) - # of students at counter: 1 and counter items P:8,C:8,D:8=24
Student 2 got food and is going to get a table (round 4) - # of empty tables: 4
Student 2 sat at table 0 to eat (round 4) - empty tables:3
Student 2 left table 0 to eat again (round 4) - empty tables:4
Student 2 is going to the counter (round 5) - # of students at counter: 1 and counter items P:7,C:7,D:7=21
Student 3 got food and is going to get a table (round 1) - # of empty tables: 4
Student 3 sat at table 0
Student 3 left table 0 to
Student 3 is going to the                                                                             6=18
Student 3 got food and is
Student 3 sat at table 0
Student 3 left table 0 to
Student 3 is going to the                                                                             5=15
Student 3 got food and is
Student 3 sat at table 0
Student 3 left table 0 to
Student 3 is going to the                                                                             4=12
Student 3 got food and is
Student 3 sat at table 0
Student 3 left table 0 to
Student 3 is going to the                                                                             3=9
Student 3 got food and is
Student 3 sat at table 0
Student 3 left table 0 to
Student 3 is going to the                                                                             2=6
Student 3 got food and is
Student 3 sat at table 0
Student 3 is done eating
Student 2 got food and is
Student 2 sat at table 0
Student 2 left table 0 to
Student 2 is going to the counter (round 6) - # of students at counter: 0 and counter items P:1,C:1,D:1=3
Student 2 got food and is going to get a table (round 6) - # of empty tables: 4
Student 2 sat at table 0 to eat (round 6) - empty tables:3
Student 2 is done eating L=6 times - going home – GOODBYE!!!
gtucpp@ubuntu:~/Desktop/midterm$
```

```
PPPPPPPPPPPPPPPDDDDDDDDDDDDDDDCCCCCCCCCCCCCCCPCDPCDPCDPCDPCDPCDPCDPCDPCDPCDPCDPCDPCDPCDPCD
```

## 3. *Testing Command Line Arguments*

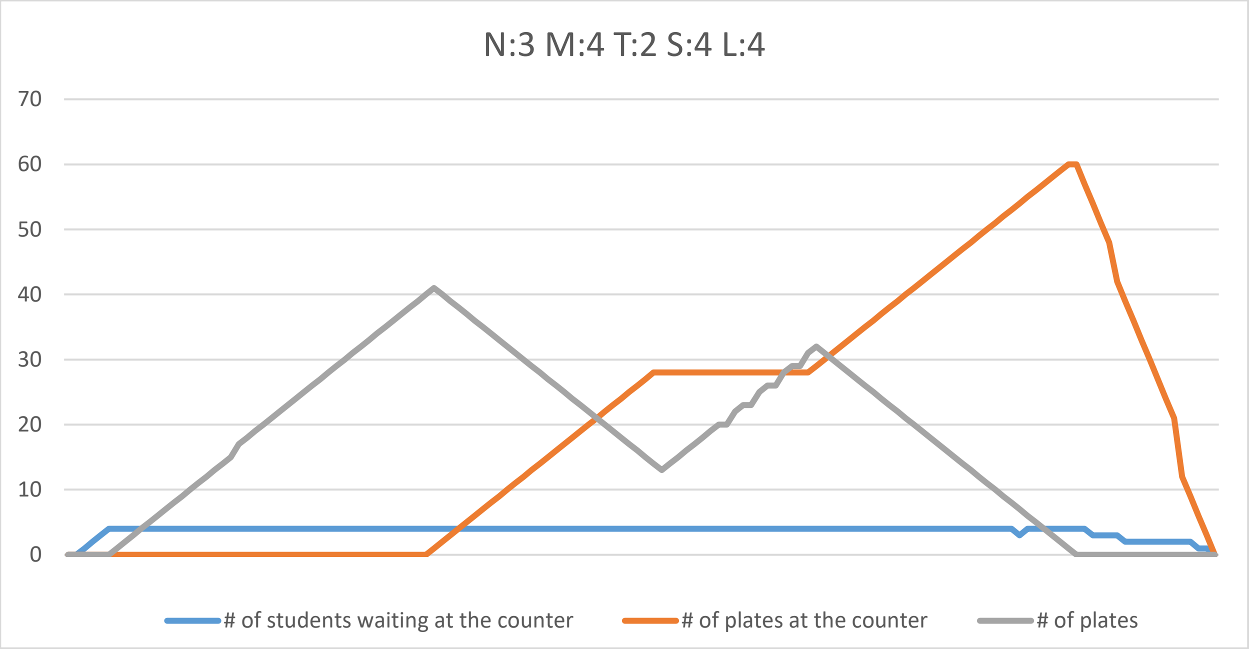


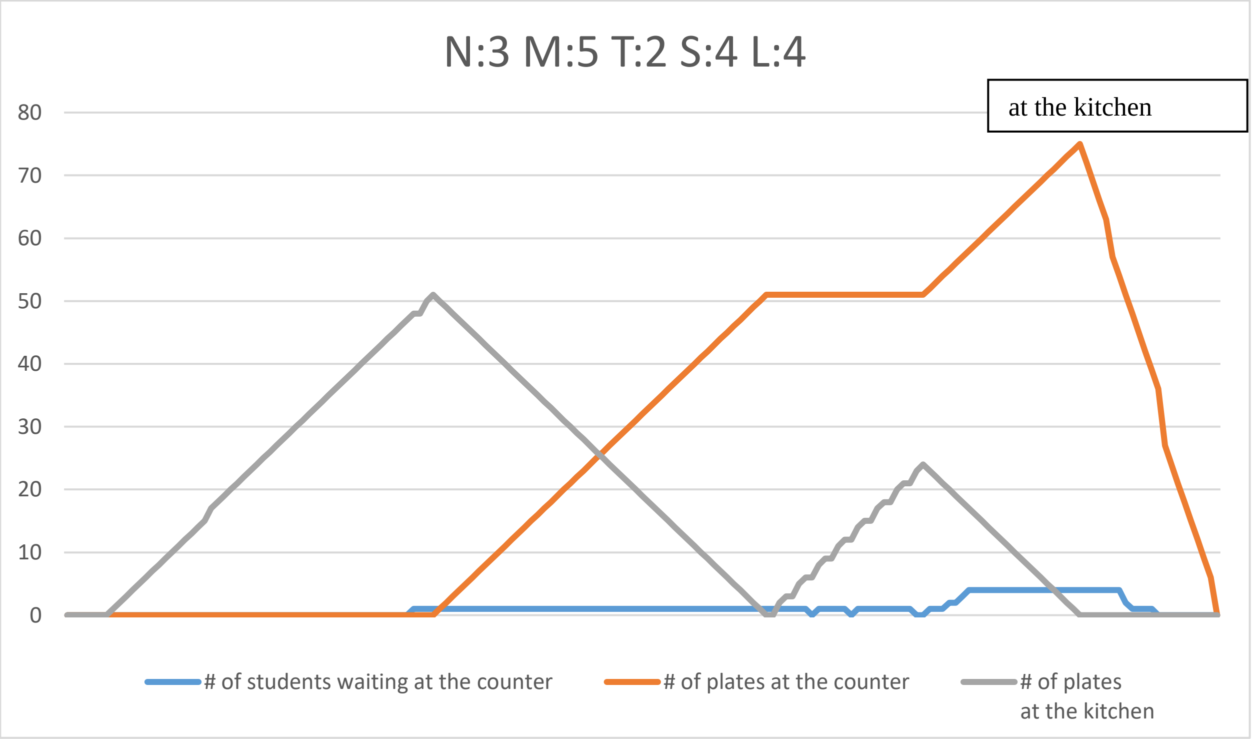
```
 -S A counter of size
 -L Every student get food from the counter, a total of L times
 -F filePath which supplier will read that must contains exactly 3LM characters
gtucpp@ubuntu:~/Desktop/midterm$ ./program -N 4 -M 3 -T 4 -S 4 -L -F filePath
Wrong command line arguments!
Usage: ./program [-NMTSL]
 -N # of cooks
 -M # of students
 -T # of tables
 -S A counter of size
 -L Every student get food from the counter, a total of L times
 -F filePath which supplier will read that must contains exactly 3LM characters
gtucpp@ubuntu:~/Desktop/midterm$ ./program -N 4 -M 3 -T 4 -S 4 -L 6 -F filePath
Wrong command line arguments!Constraints
M>N>2
S>3
M>T>=1
L>=3
gtucpp@ubuntu:~/Desktop/midterm$ ./program -N 4 -M 5 -T 4 -S 4 -L 1 -F filePath
Wrong command line arguments!Constraints
M>N>2
S>3
M>T>=1
L>=3
gtucpp@ubuntu:~/Desktop/midterm$ ./program -N 2 -M 5 -T 4 -S 4 -L 5 -F filePath
Wrong command line arguments!Constraints
M>N>2
S>3
M>T>=1
L>=3
gtucpp@ubuntu:~/Desktop/midterm$ ./program -N 2 -M 5 -T 4 -S 4 -L 5 -F filePath -X
Wrong command line arguments!
Usage: ./program [-NMTSL]
 -N # of cooks
 -M # of students
 -T # of tables
 -S A counter of size
 -L Every student get food from the counter, a total of L times
 -F filePath which supplier will read that must contains exactly 3LM characters
gtucpp@ubuntu:~/Desktop/midterm$
```

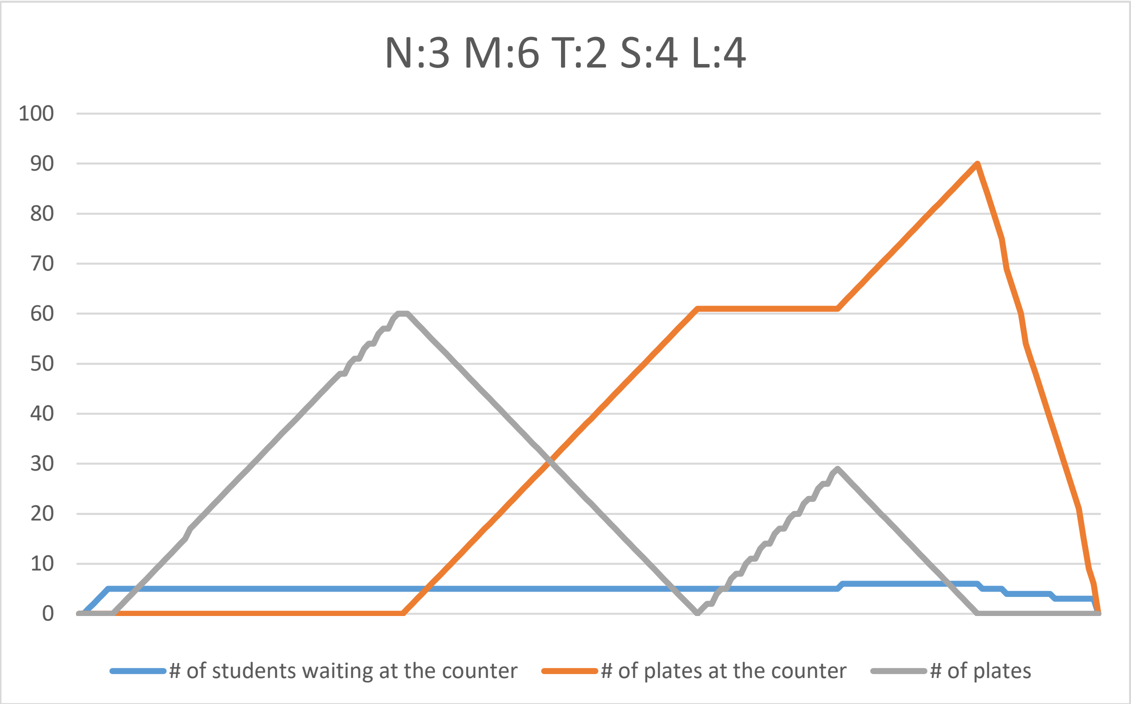
## 4. Grafics

In total, I couldn't limit the program with S. My program runs independently from S variable. So I did not drew plots.

### *1. When M is changeable and others are constant:*

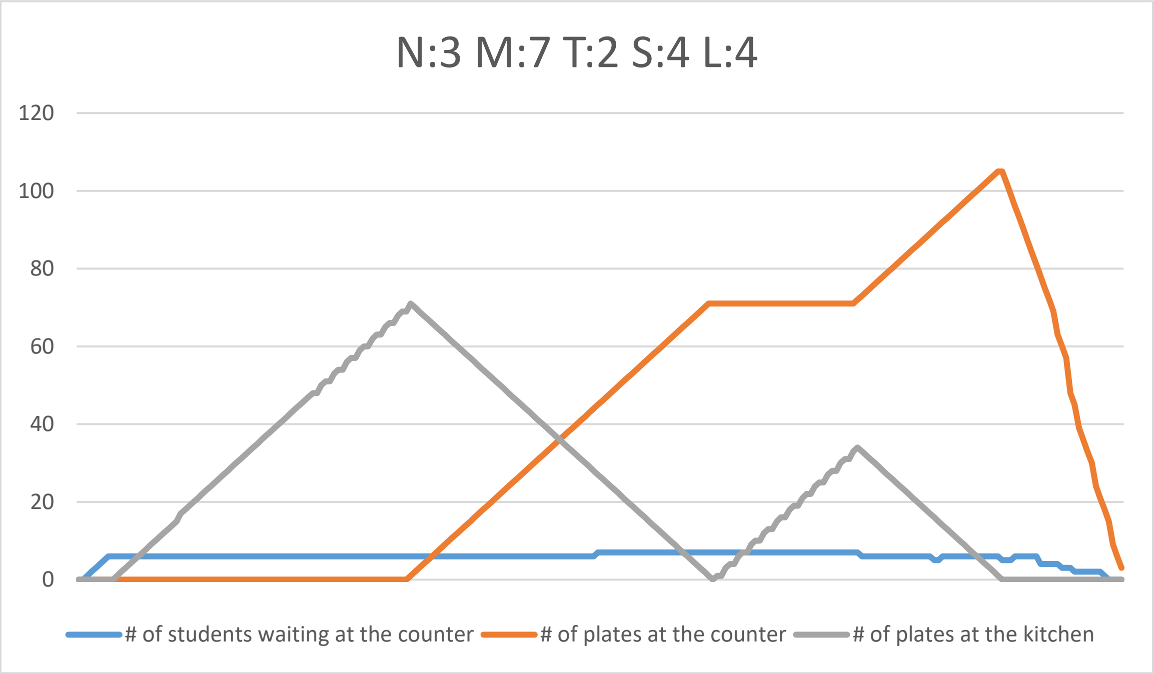


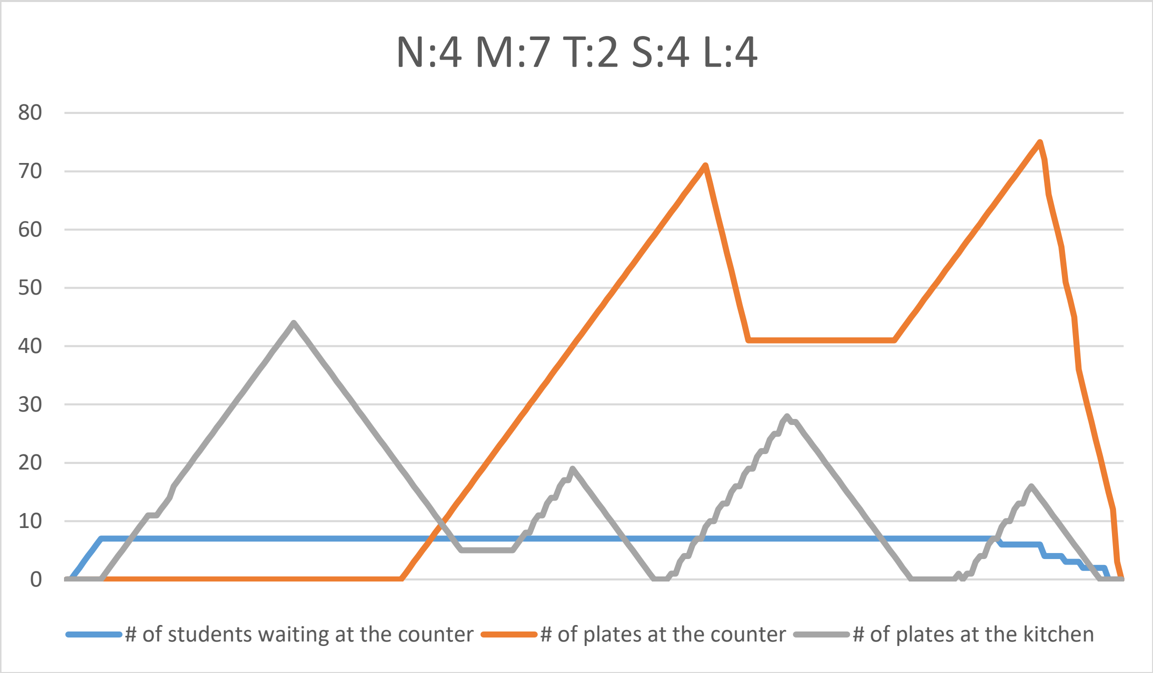


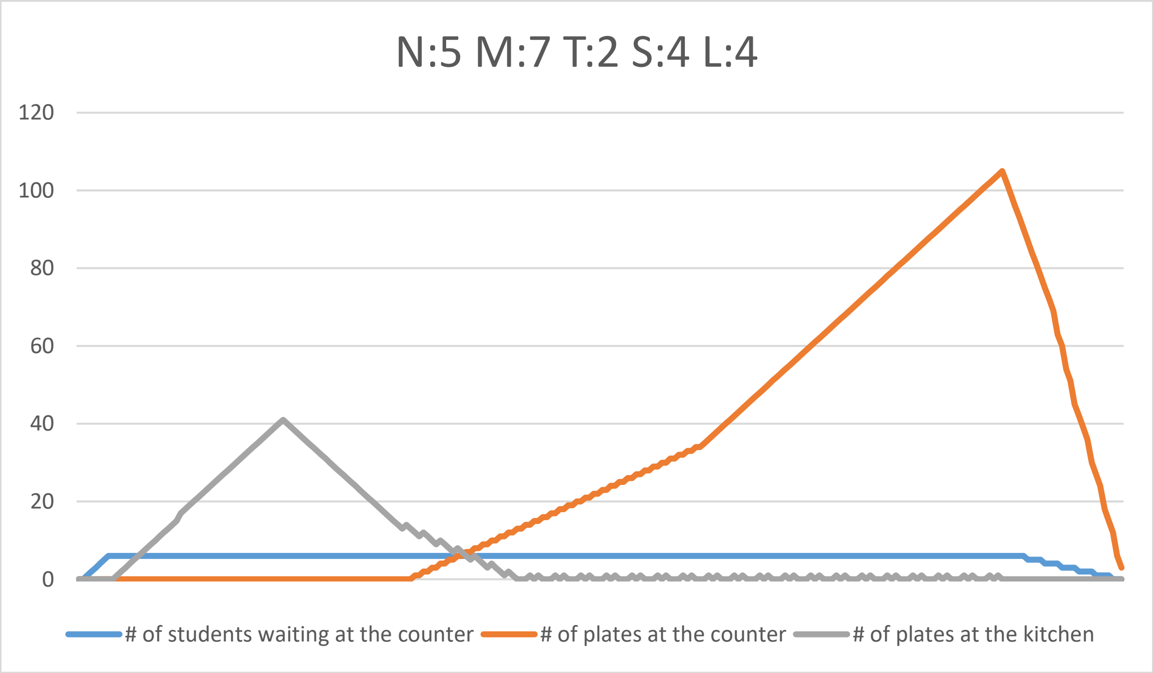


at the kitchen

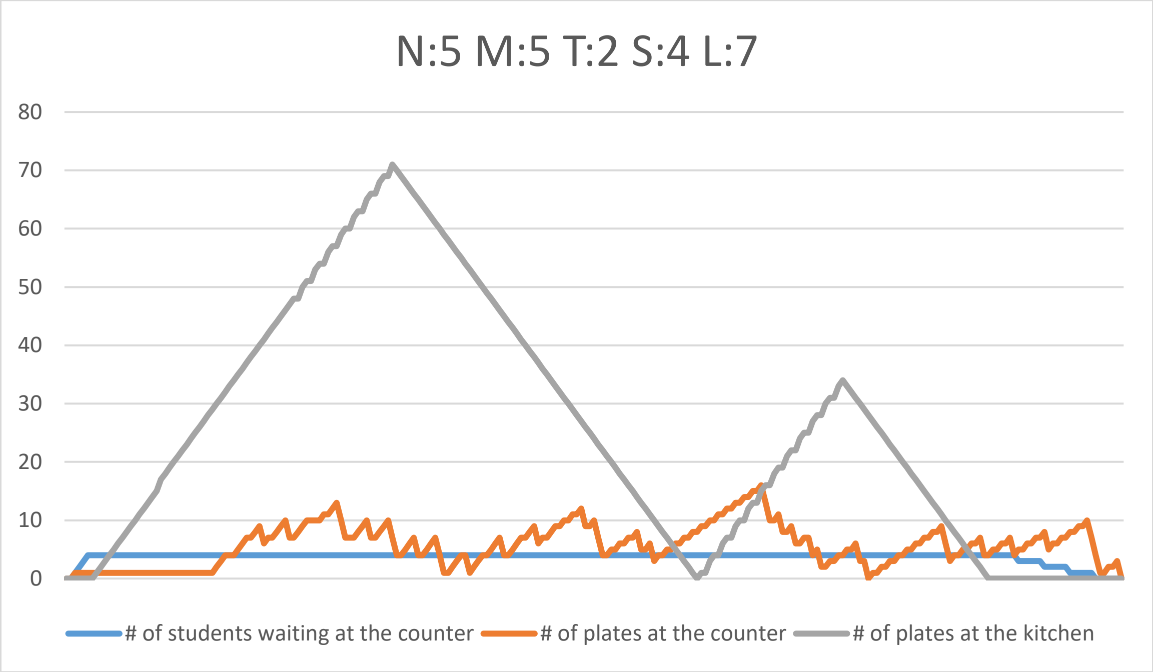
## 2. *When N is changeable and others are constant:*

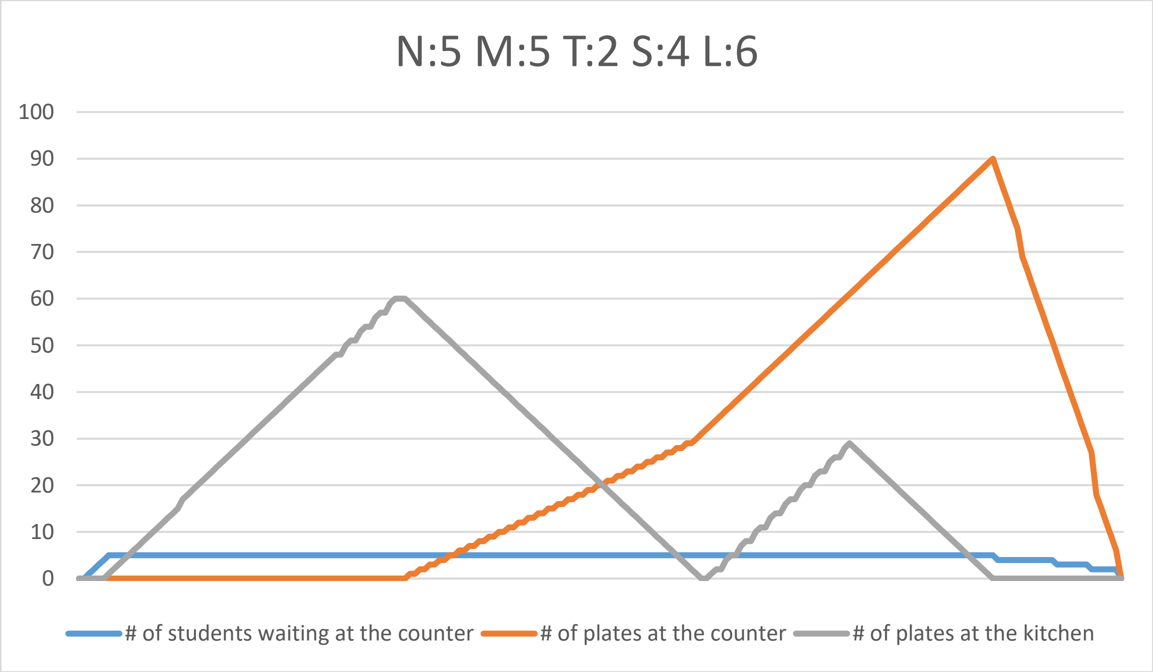


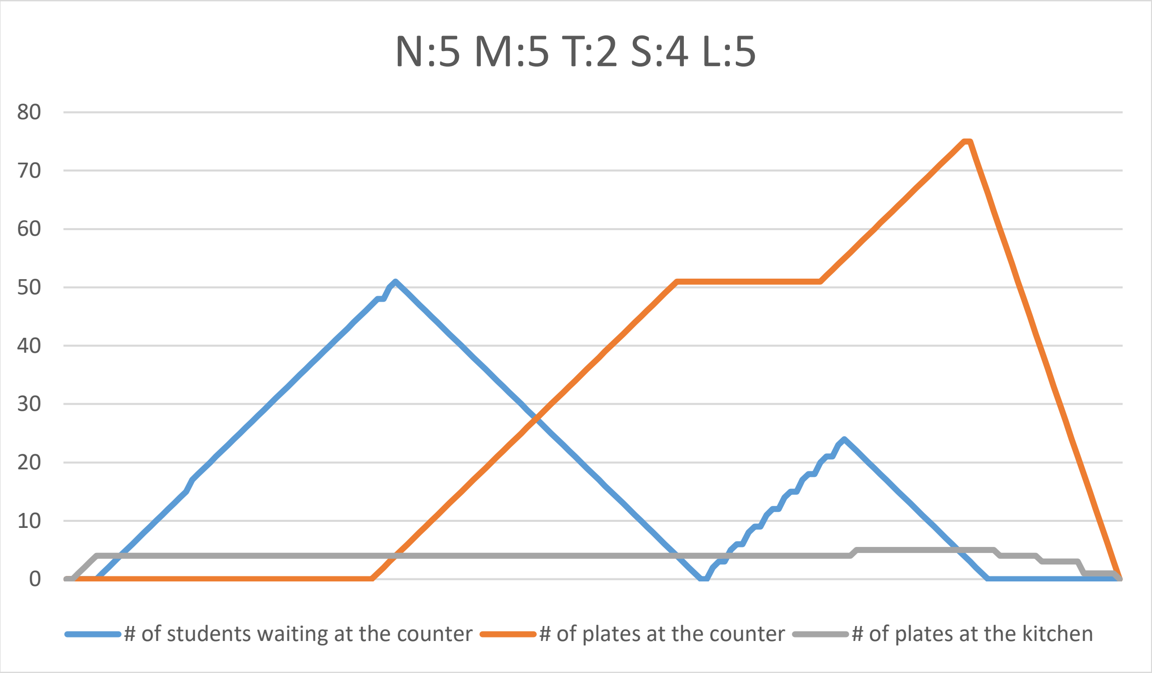

N:4 M:7 T:2 S:4 L:4
80
70
60
50
40
30
20
10
0
# of students waiting at the counter
# of plates at the counter
# of plates at the kitchen


N:5 M:7 T:2 S:4 L:4
120
100
80
60
40
20
0
# of students waiting at the counter
# of plates at the counter
# of plates at the kitchen

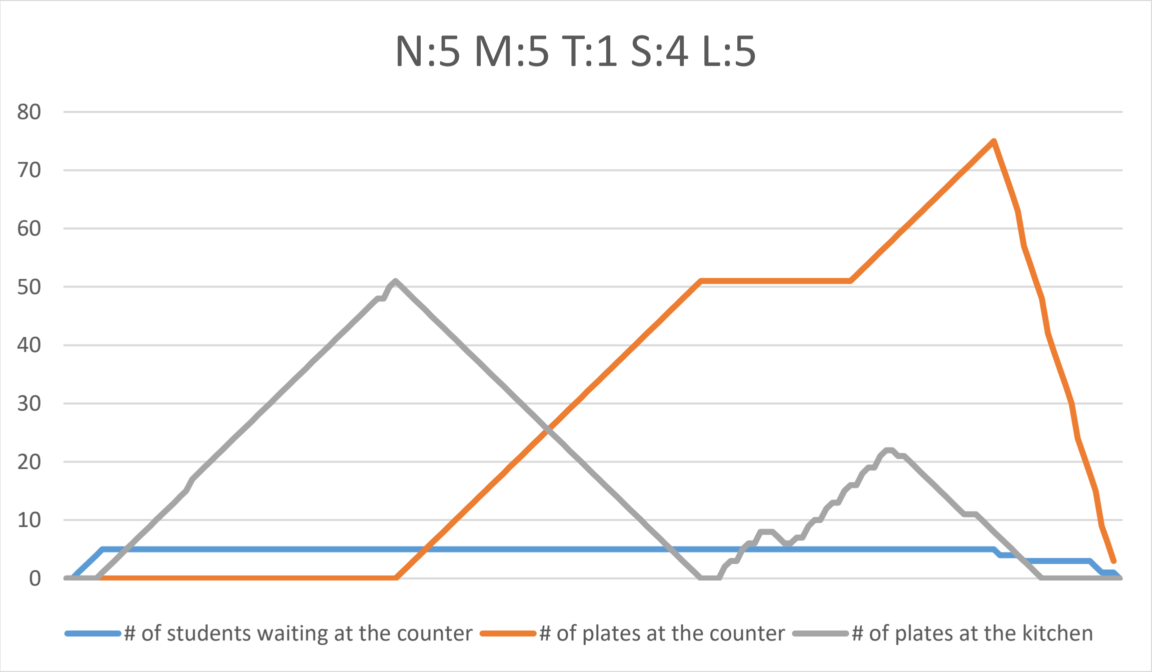
### *3. When L is changeable and others are constant:*

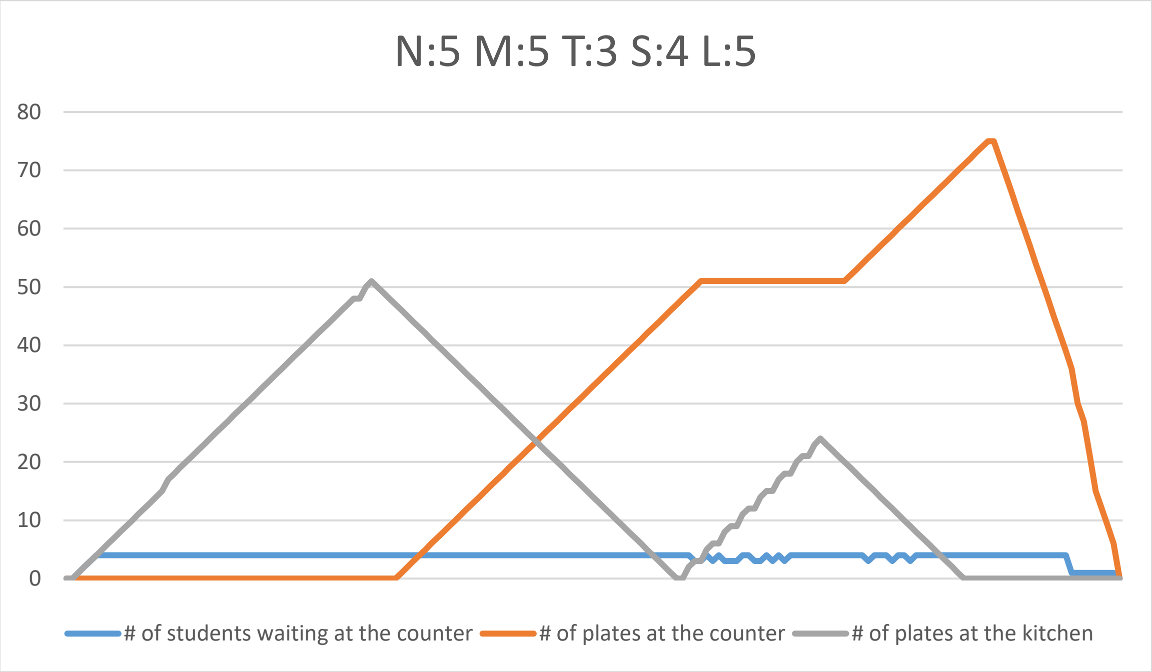


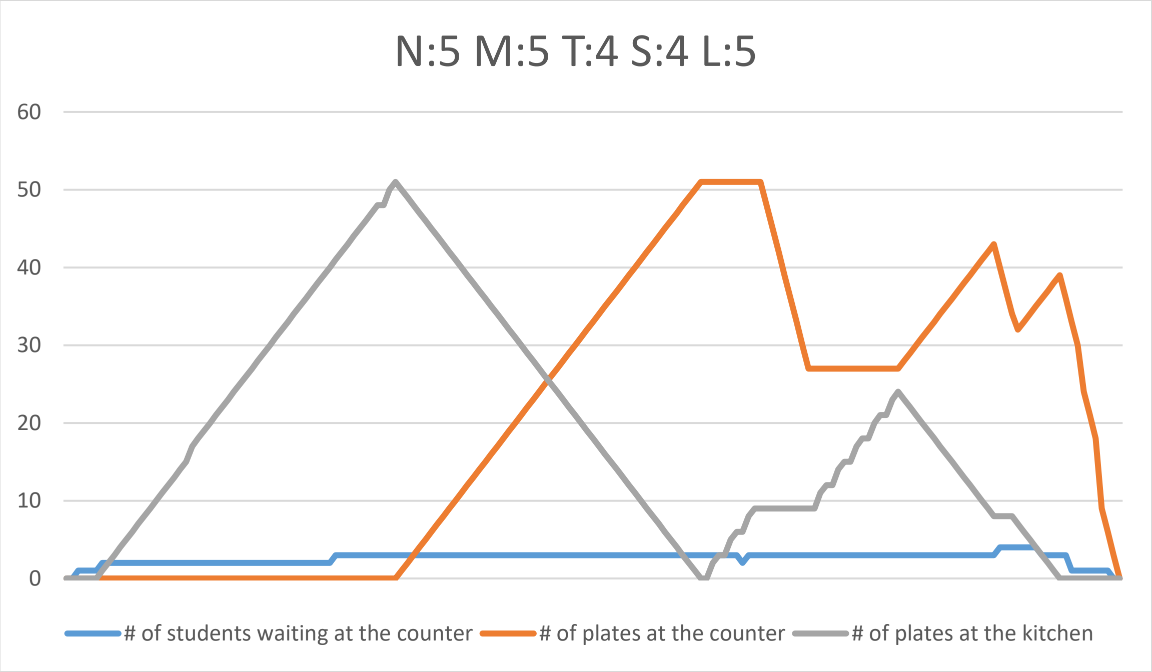




### *4. When T is changeable and others are constant:*




N:5 M:5 T:3 S:4 L:5
80
70
60
50
40
30
20
10
0
# of students waiting at the counter
# of plates at the counter
# of plates at the kitchen


N:5 M:5 T:4 S:4 L:5
60
50
40
30
20
10
0
# of students waiting at the counter
# of plates at the counter
# of plates at the kitchen