Midterm Project CSE344
System Programming
Hamza YOĞURTCUOĞLU - 171044086

→ main.c :

USAGE CHECK

I put the **Usage** at first. I seperated command line argument with **getopt().** In case of incorrect argument usage, usage is written on the screen as standard error with **write(2,.,.)** system call.

Then checked all Constraints (all are integers)

M > N > 2

S > 3

M > T >= 1

L >= 3

K = 2LM+1

U + N + M < 10000

When I run the above 10000 programs(N+U+G). Linux is crashing, I would like to state that this situation is not related to the program. This can vary from computer to computer. When something is written outside of these limits, it is written on the screen with standard error.

It is checked whether there are 3ML characters in the file and there are equal number of 'P', 'C' and 'D' characters. If there are not enough characters or there are no equal numbers 'P', 'C' and 'D' characters, an error was written on the screen with standtard error.

SHARED MAPPING

In this section, it will be explained why the shared semaphores, variables and arrays received are used.

SIZE = 16 - SEMAPHORE (8 MUTEX + 8 SEMAPHORE) ARRAY

SIZE = 18 - INT SHARED VARIABLE ARRAY

SIZE = T(TABLE SIZE) - INT SHARED ARRAY

3 SHARED ARRAY (That is used like queue in cookers for foods.)

SEMS 0 → Kitchen full semaphore. When Supplier adds any item, this semaphore will be decrease by supplier. Cooker needs to increase this semaphore to receive items.

SEMS 1,2,3 → All three semaphores were kept in the kitchen for soup, main course and desert respectively. Supplier increases the semaphore by 1, so that the cooks can take it. If they need them.

SEMS 4 → Kitchen empty semaphore. When Supplier adds any item, this semaphore will be increased by supplier. Cooker needs to reduce this semaphore to receive items.

SEMS 5 → Counter full semaphore. When any cooker adds any item, this semaphore will be decreased by cooker. Cooker needs to reduce this semaphore to receive items.

SEMS 6 → Critical section mutex for cookers (thus there is no starvation). I researched starvation of process. I implemented like **LittleBookOfSemaphores** (85 page). Mutex required to create two sections between cookers.

SEMS 7 → Mutex for the number of items in the kitchen between supplier and cookers

SEMS 8 → Second mutex for cooker critic section 2 , first section is sem6 mutex.

SEMS 9 → Table full semaphore. This semaphore should be reduced when any student tries to get a table. If this semaphore is 0, the student will wait.

SEMS 10 → Table variable mutex. Each table has an index. The student should reduce this mutex while changing an empty place in the table array.

SEMS 11 → Ready service number for student. a ready service (at least one soup, one main course and one desert are available on the counter) it sends a signal to the student via this semaphore in order to **take all 3 of them at once**. If there is a **graduate**, his semaphore is increased. **(SEM 15)**

SEMS 12 → counter item mutex. When a change is made in the counter, this mutex should be taken.

SEMS 13 → **N**umber of students at counter mutex.

SEMS 14 → **M**utex between graduate and cooker. Through this mutex, the number of graduate students waiting in the counter can reach the cooker correctly.

SEMS 15 → Ready service number for **graduated student.** If there is a graduated student waiting at the counter, when the cooker service (at least one soup, one main course and one desert are available on the counter) is ready, the cooker sends this semaphore signal instead of sending the signal **SEM 11** so that the graduate student can get his meal in order to **take all 3 of them at once**. We see that graduate students have priority at the counter over undergraduates.

SEMS 16 → In order to print console mutex that is common for all processes.

VARS 0,1,2 → All three variables were kept in the kitchen for soup, main course and desert respectively. Supplier increases the variable by 1, so that the cooks can take it. If they need them. We print screen this numbers.

VARS 3 → Number of items in this kitchen.

VARS 4 → All the cooks will be out when how many meals Cookers left on the counter when 3ML.

VARS 5,6,7 → All three variables were kept in the counter for soup, main course and desert respectively. Cookers increases the variable by 1, so that the student can take at least one soup, one main course and one desert are available on the counter. If they need them. We print screen this numbers.

VARS 8 → Supplier is over or not. This variable is for cooker. If the last 3 items are left in the kitchen and the supplier duty is over, the cookers take the items directly and place them in the counter.

VARS 9 → Number of empty tables.

VARS 10,11,12,13,14,15 → The back and front indexes of the queues needed to understand if there is a service for COOKERS.

10 → soup queue rear. 11 → soup queue front. 12 … respectively

VARS 16 → Number of students at counter

VARS 17 → Number of **graduated** students at counter

tableNum → All tables are assigned first empty(0) - full table(1). Each table has been indexed through this array.

soupQ, mainQ and desertQ → These queues are made to understand if there is a service. **Example : P:2,C:1,D:0=3**

If cooker add a desert in desertQ and cooker will check what there is a service or not.

P:2,C:1,D:1=4.

soupQ : 1 1

mainQ : 1 We understand that there is one service for student.

desertQ :1

Then delete the a 1's each queues and send signal graduate student or student in order to take service.

supplier.c : After checking the number of characters in the file, it is left to the supplier reading. According to the character read from the supplier, if the kitchen is not full, it increases the soup, main course or desert semophore.

cooks.c :

Every cook is checking which food the counter needs. Depending on the state of the counter, cooker goes to the kitchen and takes that plate. The purpose of the cookers is to always create service and try to fill the counter until full.

When each cooker puts a plate on the counter, it increases the common variable (vars [4]) by 1 in all cooker. In this way, all the cooks understand that all the plates(**3*M*L**) are placed in the counter and the cycle ends.

→ HOW DO COOKER UNDERSTAND THAT THERE IS AT LEAST 1 SERVICE?

Each plate (soup, main course and desert) queues are available to understand that there is a service.

Example : P:2,C:1,D:0=3

If cooker add a desert in desertQ and cooker will check what there is a service or not.

P:2,C:1,D:1=4.

soupQ : 1 1

mainQ : 1 We understand that there is one service for student.

desertQ :1

Then delete the a 1's each queues and send signal graduate student or student in order to take service.

Then deleted the 1's like following

P:1,C:0,D:0=1.

soupQ : 1

mainQ : - We see that each front of queues must 1 in order to send a signal

desertQ : -

→ HOW DO COOKER UNDERSTAND THAT THERE IS GRADUATED STUDENT AT COUNTER?

When a cooker service is ready, vars [17] (grad. Student count on the counter) checks. If there are [17] greater than zero, the graduate student semaphore**(sem[15])** is posted.

studentandgraduate.c : While creating student processes, **gradOrStudent** variable is given as argument. If this variable is **1** that means this process s graduated student,**0** is only student.

When students arrive at the counter, it increases the number of students waiting at the counter(**var[16]**). Graduate students increase the number of grad. students waiting at their counter(**var[17]**). In this way, if there is a ready service, the service is given **directly to the graduated student**.

HOW DO STUDENTS OR GRADS. SIT A TABLE?

All student and grad. a common table array was created. First, this array was initialized with 0 (all table are empty). Every student and grad. If he take their service, they will check the array from the first element until they saw 0. If one sees 0, it decreases the number of empty tables by one and makes the element of the array in index 1 (indexed table full).

When leaving, it increases the number of empty tables by one, and makes the value of the indexed table array 0.

Example :

Empty Table : 5 Table Array : 0 0 0 0 0

When student or grad. come to in table.

Empty Table : 4 Table Array : 1 0 0 0 0 Table 1 is taken

When student or grad. come to in table.

Empty Table : 3 Table Array : 1 1 0 0 0 Table 2 is taken

…

When student or grad. come to in table.

Empty Table : 0 Table Array : 1 1 1 1 1 They will wait until any table will be free

Finally, when the student or grad. complete the last round (L), they exit the loop.

WHEN CTRL + C PRESSING IS DONE, ARE ALL THE RESOURCES BEIGN RELEASED ?

Pointers of all the resources received were defined globally. In this way, if a memory allocate is made from a method, these resources can be released in SIGINT(**CTRL+C**) handler. Below you can see the result of **valgrind**:

```
==11999== HEAP SUMMARY:
==11999==     in use at exit: 0 bytes in 0 blocks
==11999==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==11999==
==11999== All heap blocks were freed -- no leaks are possible
==11999==
==11999== For counts of detected and suppressed errors, rerun with: -v
==11999== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 1 : Freed all resources when ctrl+c pressing and normal execution.

There is no error or warning from any processes.(This image is just a process)

In addition, the main process is waiting in ctrl + c for the children to finish in its handler method. In this way, **there are no zombies.**

→ Test Program Examples: The following commands write how you can run process. I tested 'N+U+G' up to 10000 processes. My computer freezes in 10000 or more processes.

You can execute with Absolute path or relative path for files.

./program -N 3 -T 5 -S 4 -L 13 -U 8 -G 2 -F filePath
./program -N 3 -T 5 -S 4 -L 13 -U 8 -G 2 -F (AbsolutePath)/filePath

→ filePath Examples: which must contain exactly 3*L*M characters, each being either 'P','C' or 'D' in an arbitrary order. P,C and D must be equal.

```
PCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPC
DCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCD
CDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDCDPDCPPCDC
```

NOTE : Dont put '/n' char or something.

→ Output Example :

```
The supplier delivered desert - after delivery: kitchen items P:54,C:55,D:55=164
The supplier is going to the kitchen to deliver soup : kitchen items P:54,C:55,D:55=164
The supplier delivered soup - after delivery: kitchen items P:55,C:55,D:55=165
Graduate Student 0 got food and is going to get a table (round 1) - # of empty tables: 5
Graduate Student 0 sat at table 1 to eat (round 1) - empty tables:4
Graduate Student 0 left (round 1) - # of empty tables: 5
Graduate Student 0 is going to the counter (round 2) - # of graduate students at counter: 2 and counter items P:0,C:0,D:0=0
The supplier is going to the kitchen to deliver desert : kitchen items P:55,C:55,D:55=165
```

Figure 2: There is lots of student at counter. But Grad.Student takes service

…

```
Student 2 is going to the counter (round 3) - # of students at counter: 8 and counter items P:1,C:1,D:1=3
Student 2 got food and is going to get a table (round 3) - # of empty tables: 5
Student 2 sat at table 1 to eat (round 3) - empty tables:4
Student 2 left (round 3) - # of empty tables: 5
```

Figure 3: If there is no grad. at counter . Normal student take service if exist service

…

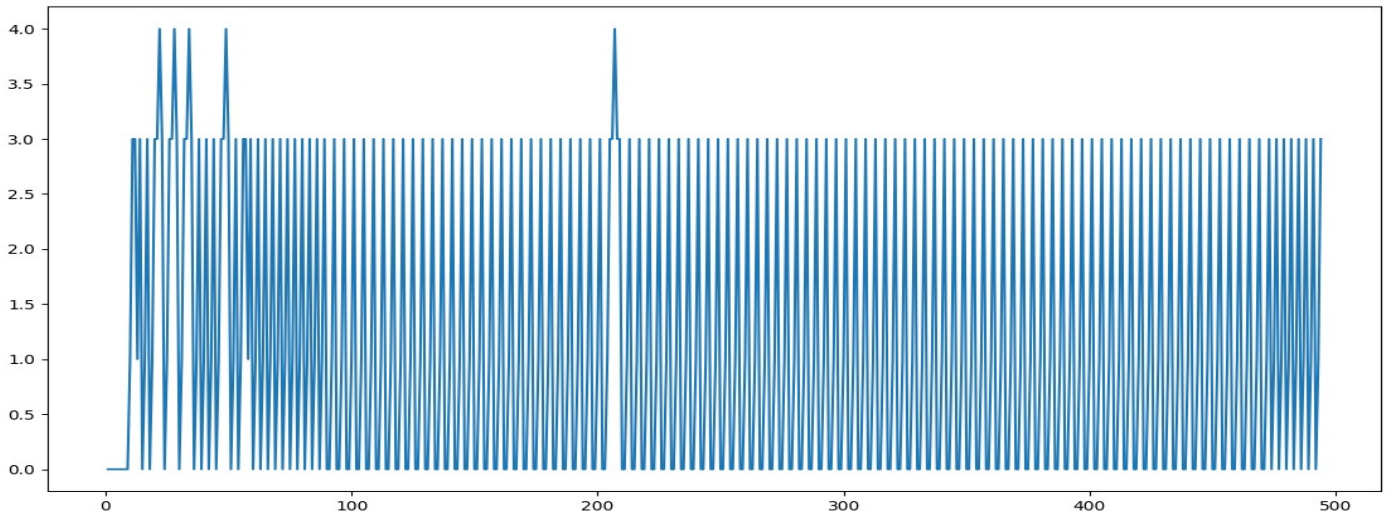
```
Student 6 got food and is going to get a table (round 13) - # of empty tables: 5
Student 6 sat at table 1 to eat (round 13) - empty tables:4
Student 6 left (round 13) - # of empty tables: 5
Student 6 is done eating L=13 times - going home - GOODBYE!!!
Cook 2 finished serving - items at kitchen: 0 - going home - GOODBYE!!!
Cook 1 finished serving - items at kitchen: 0 - going home - GOODBYE!!!
Cook 0 finished serving - items at kitchen: 0 - going home - GOODBYE!!!
```

Figure 4: Finished all processes.
This example is : ./program -N 3 -T 5 -S 4 -L 13 -U 8 -G 2 -F file

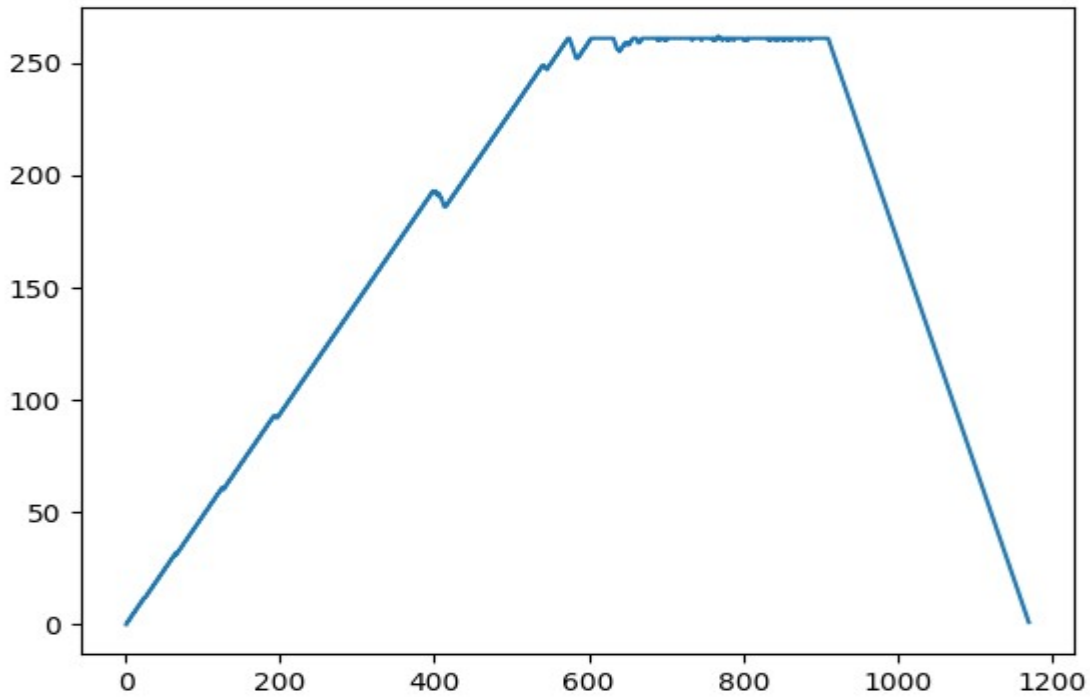
Graphical Plots

In order to draw all graphical plots, the numbers were made to the file first. Then plots were drawn with the matplotlib.pyplot library in python.

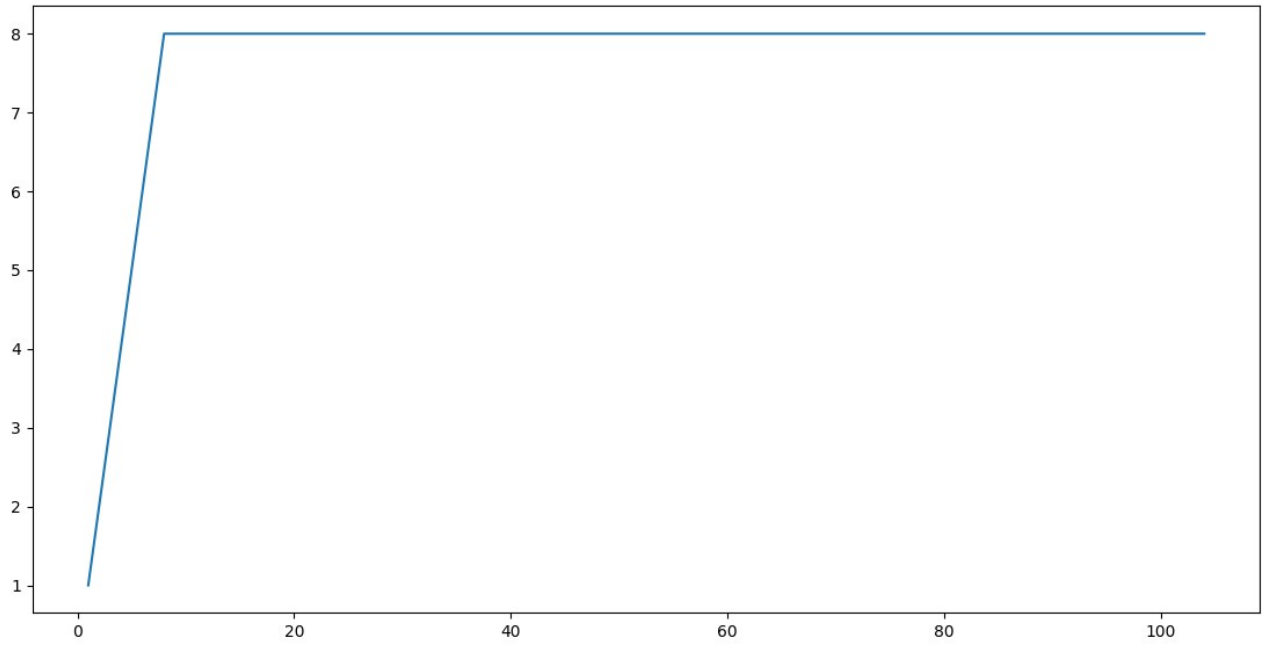
./program -N 3 -T 5 -S 4 -L 13 -U 8 -G 2 -F file



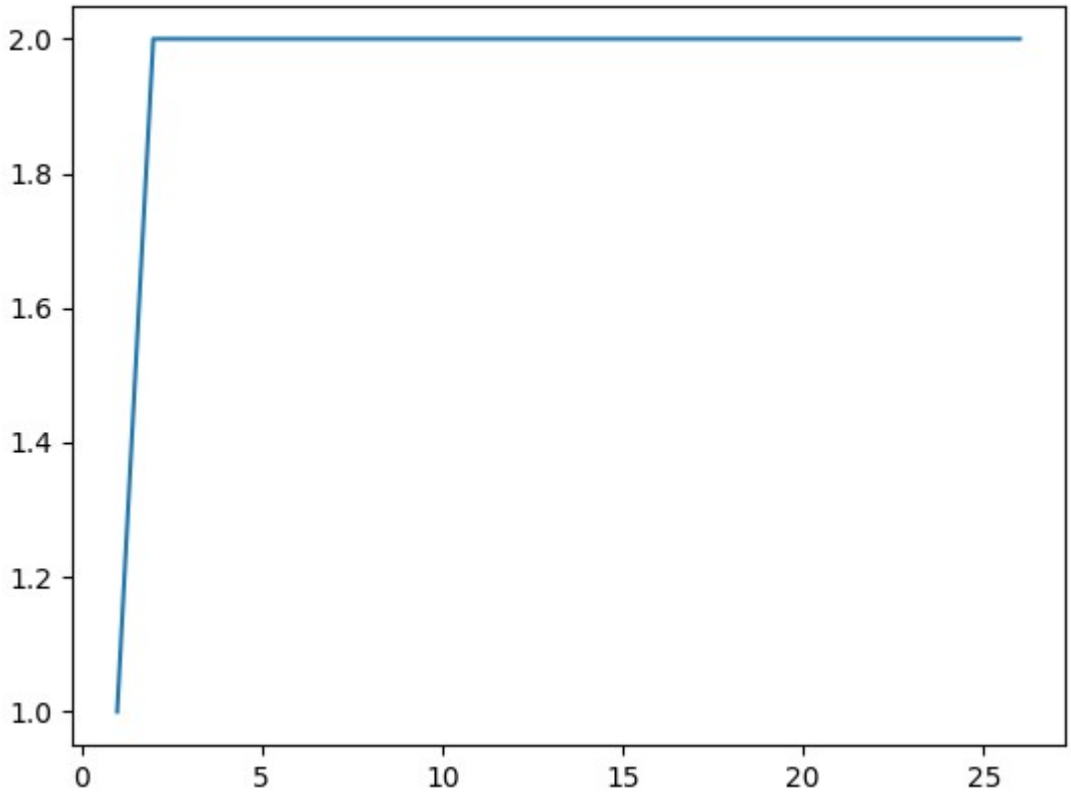
Plates at counter



Plates at kitchen

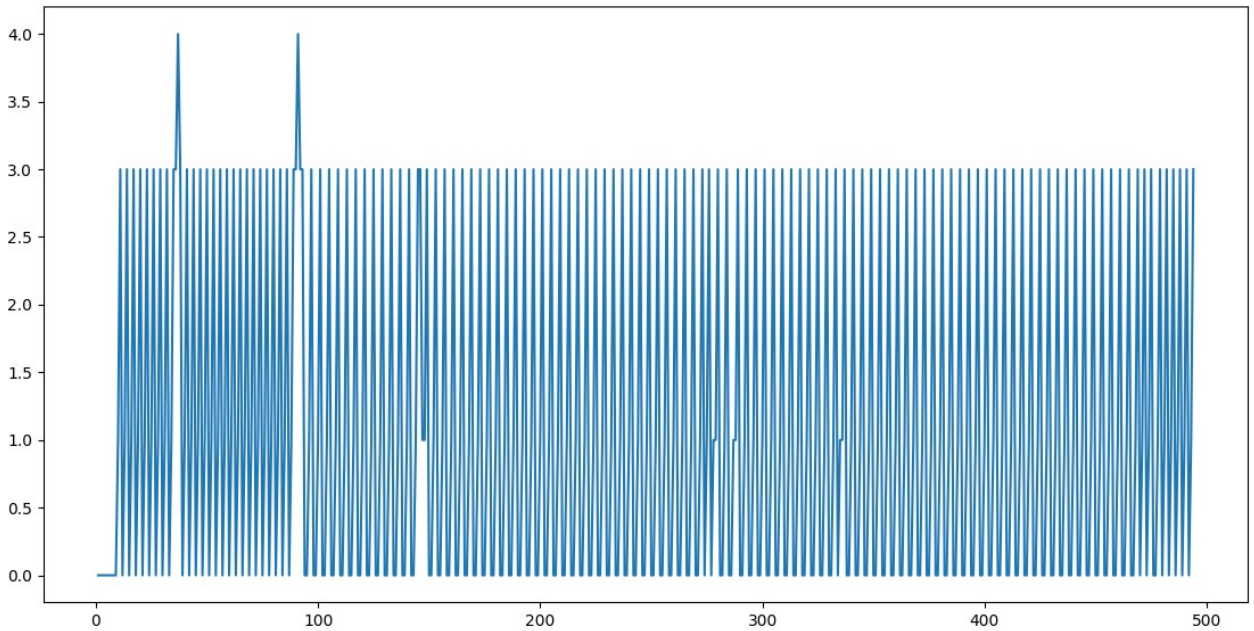


Students at counter

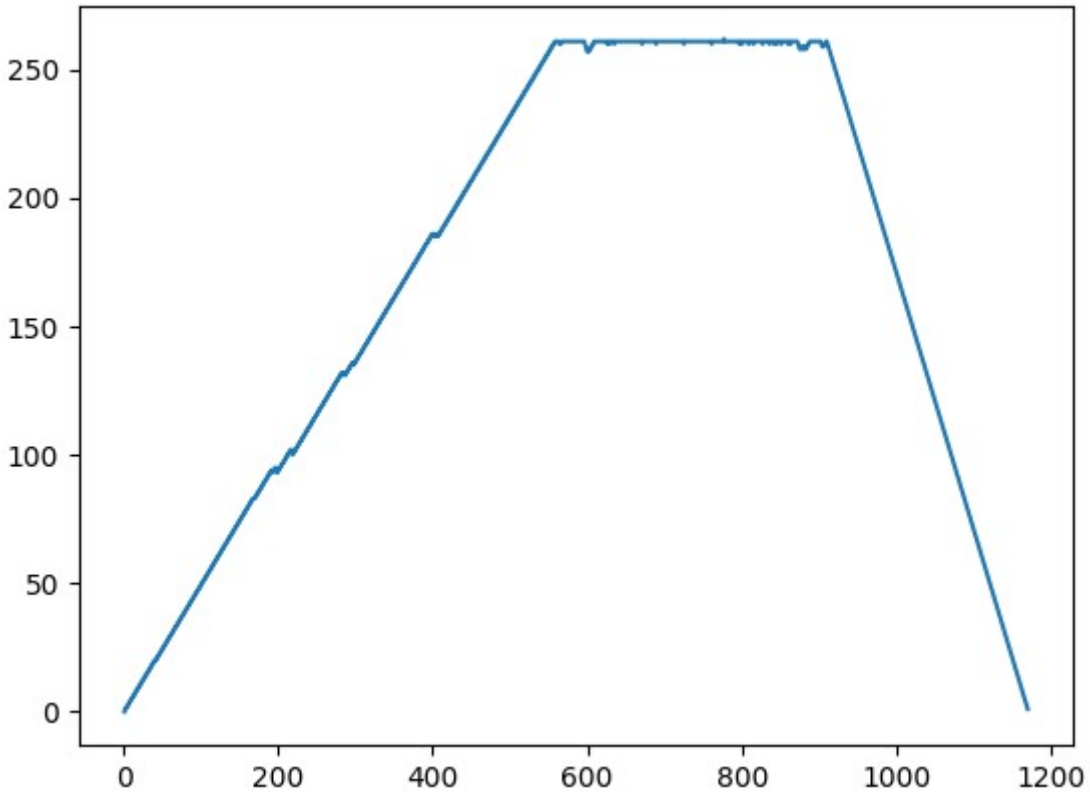


Graduated Student at Counter

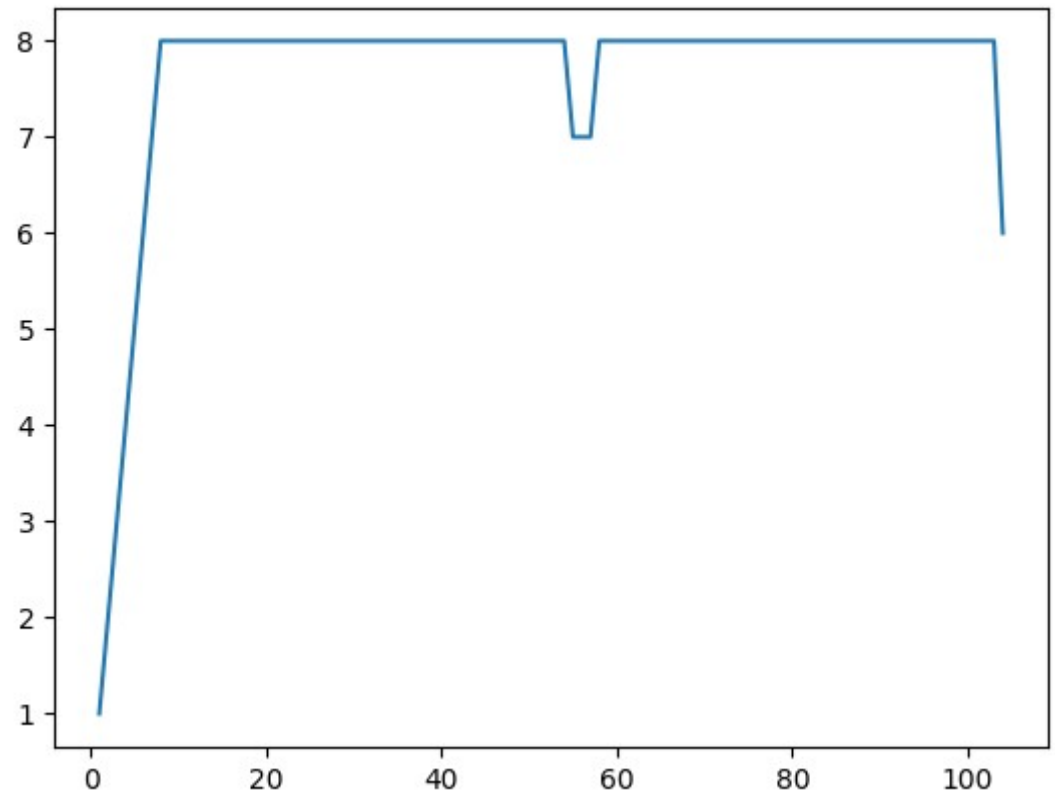
./program -N 6 -T 5 -S 4 -L 13 -U 8 -G 2 -F file



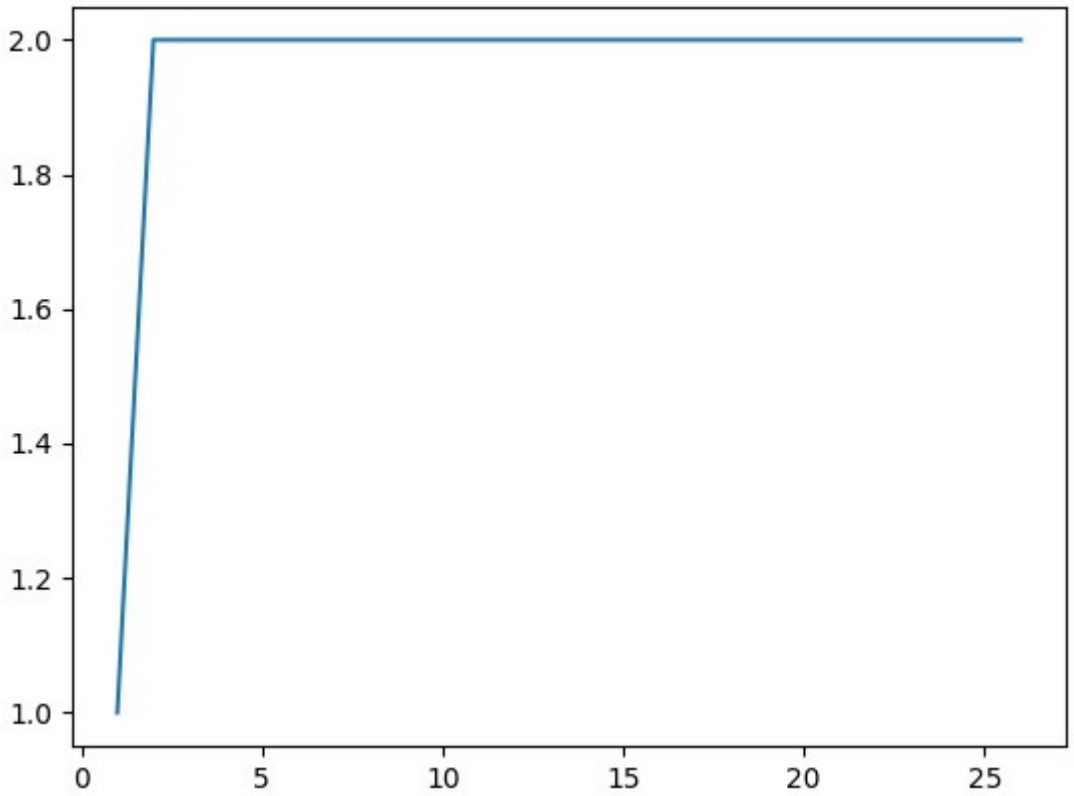
Plates at counter



Plates at kitchen

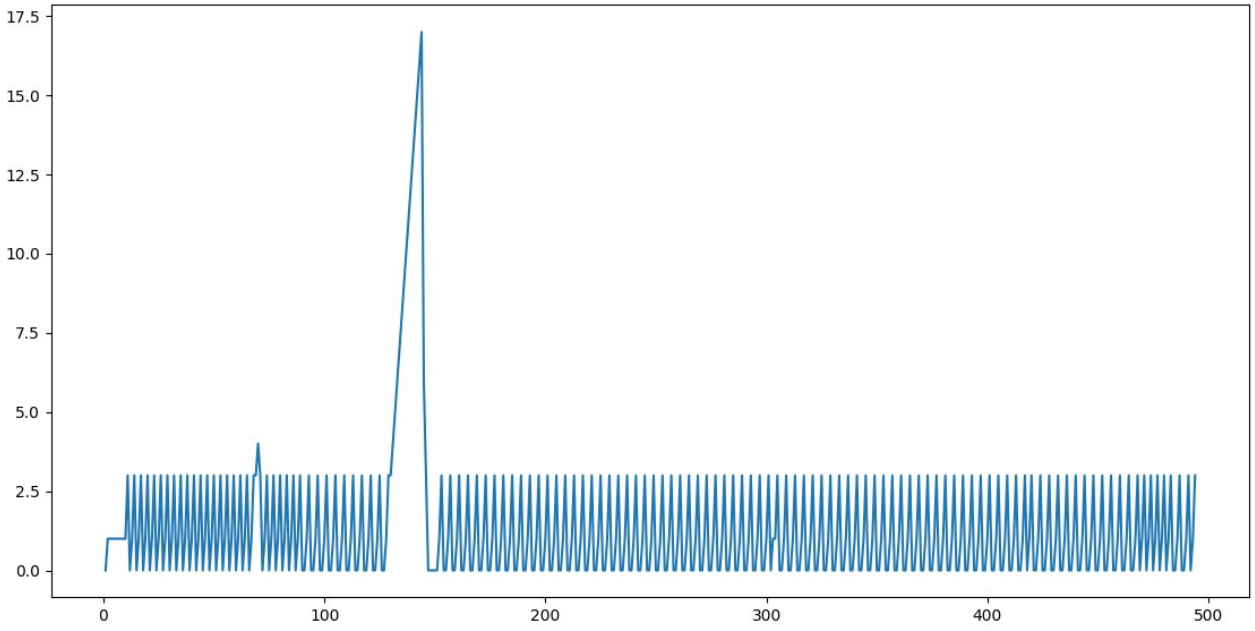


Students at counter

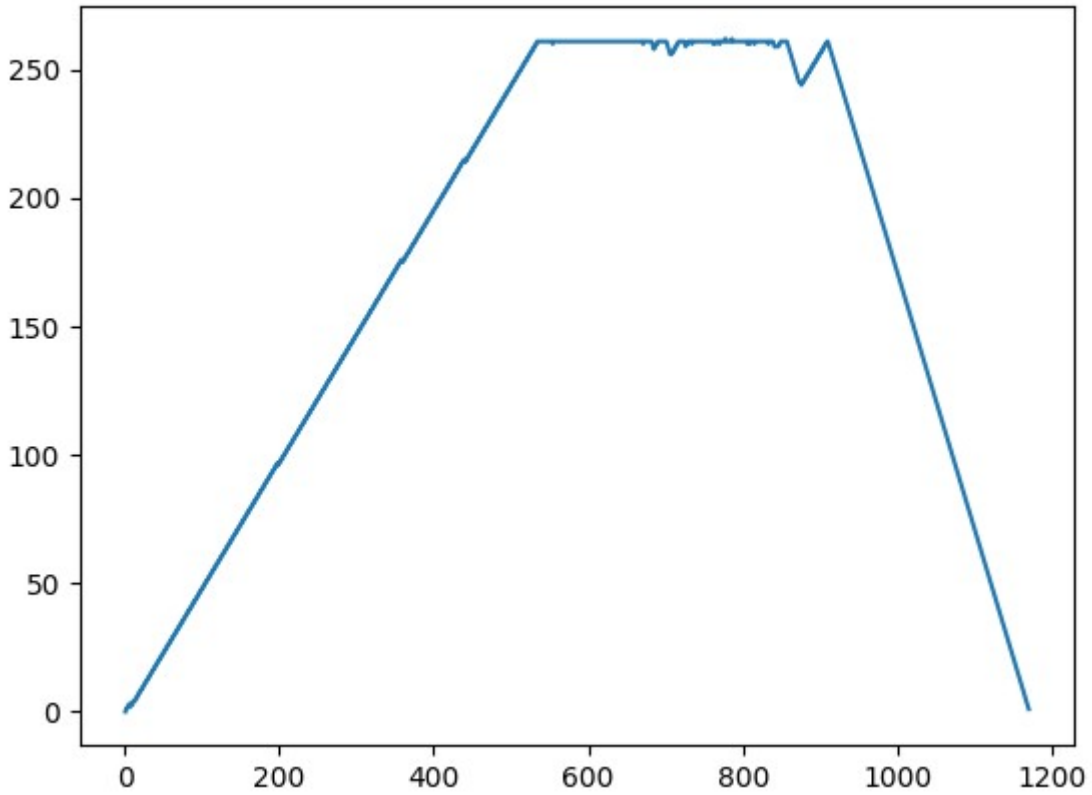


Graduated Student at Counter

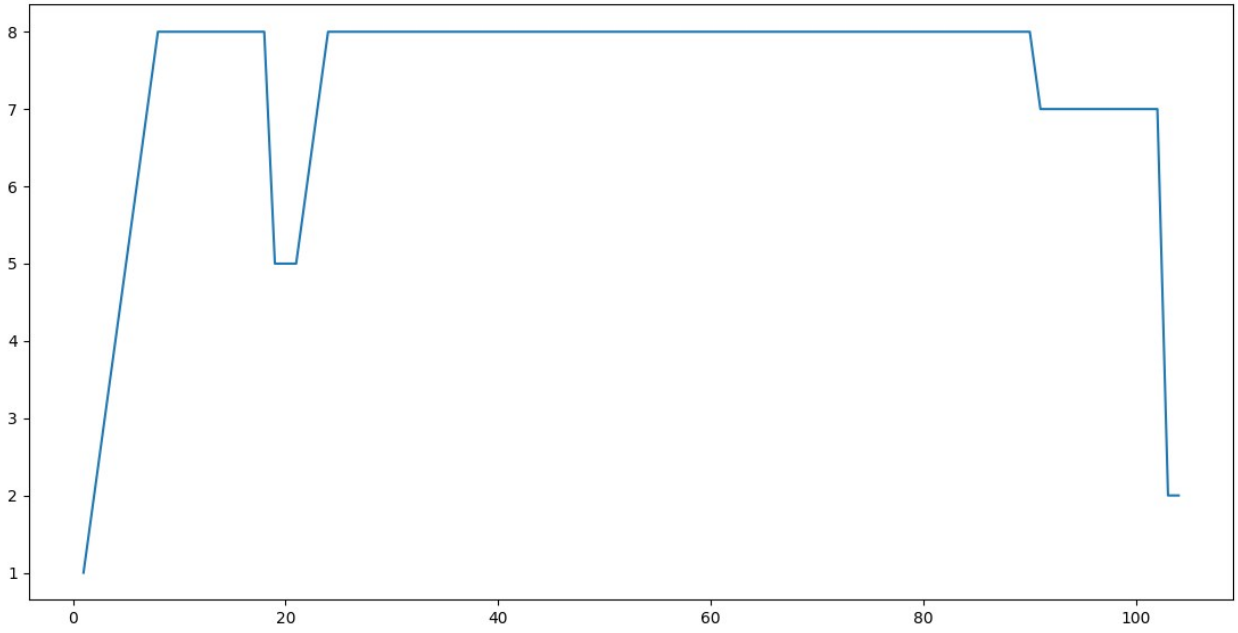
./program -N 6 -T 5 -S 40 -L 13 -U 8 -G 2 -F file



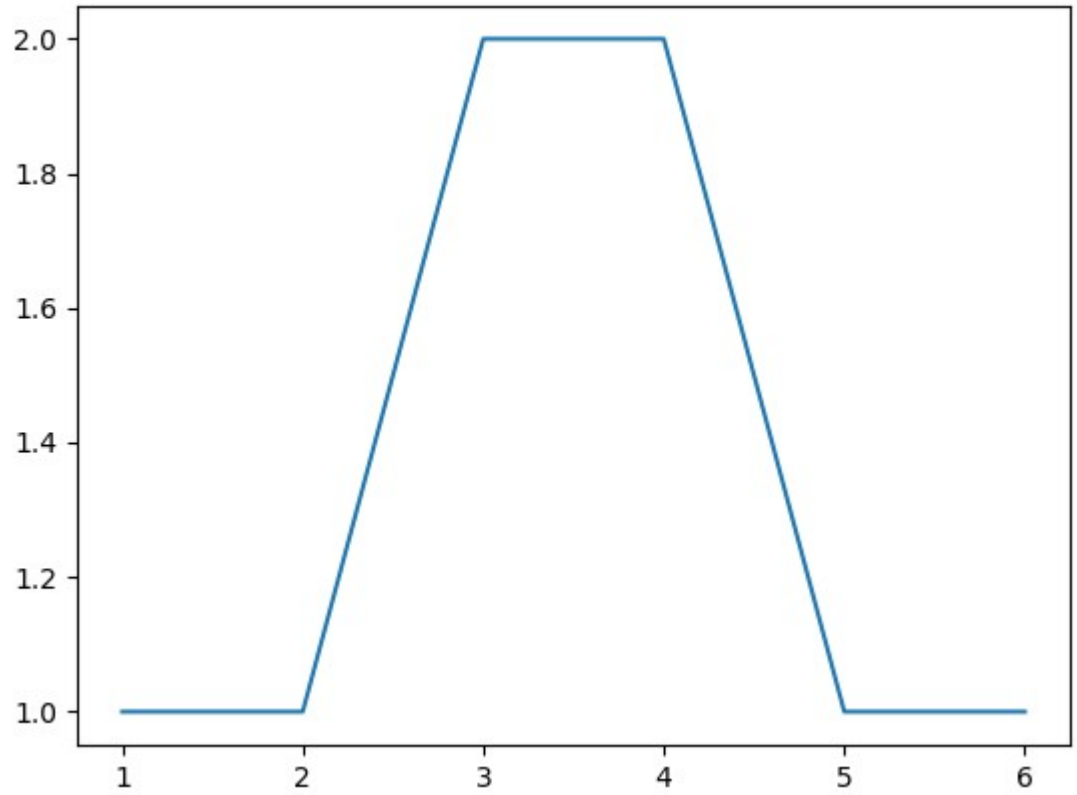
Plates at counter



Plates at kitchen

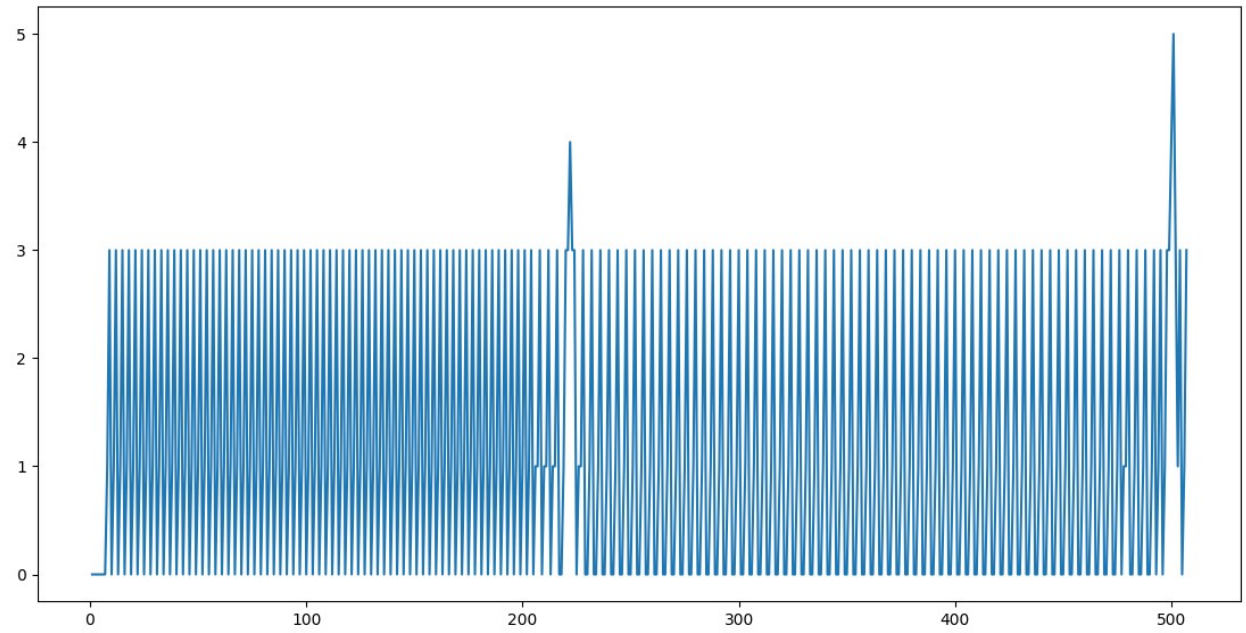


Students at counter

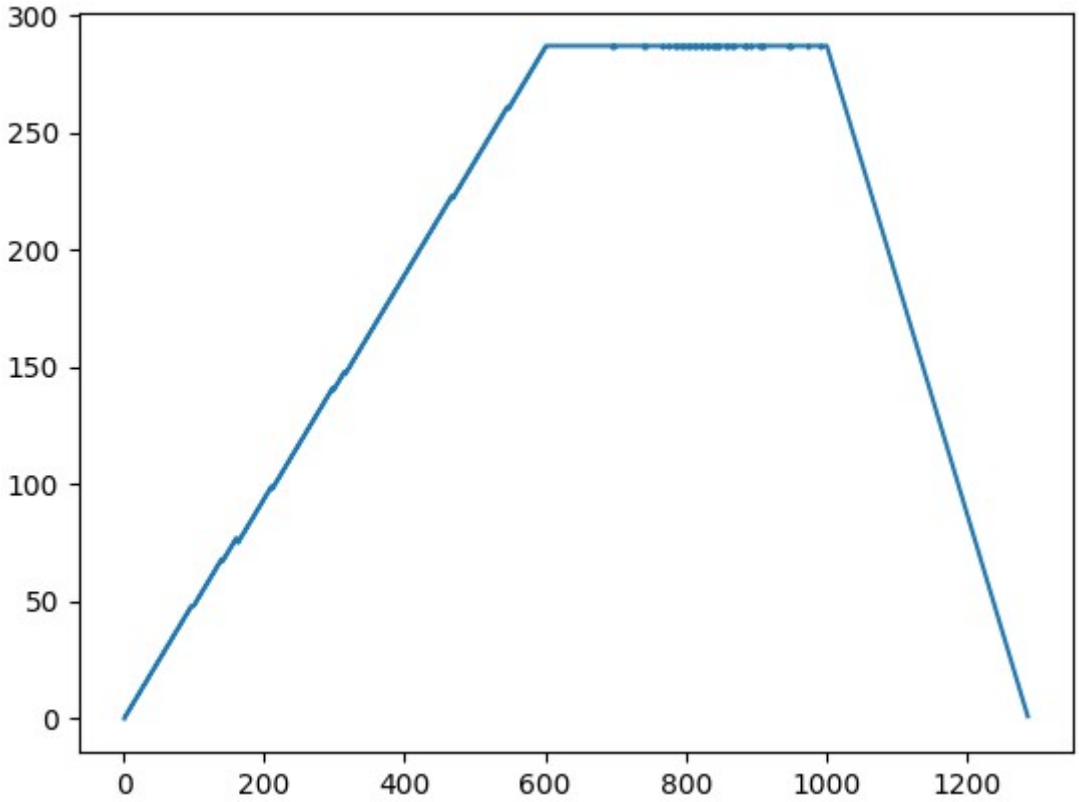


Graduated Student at Counter

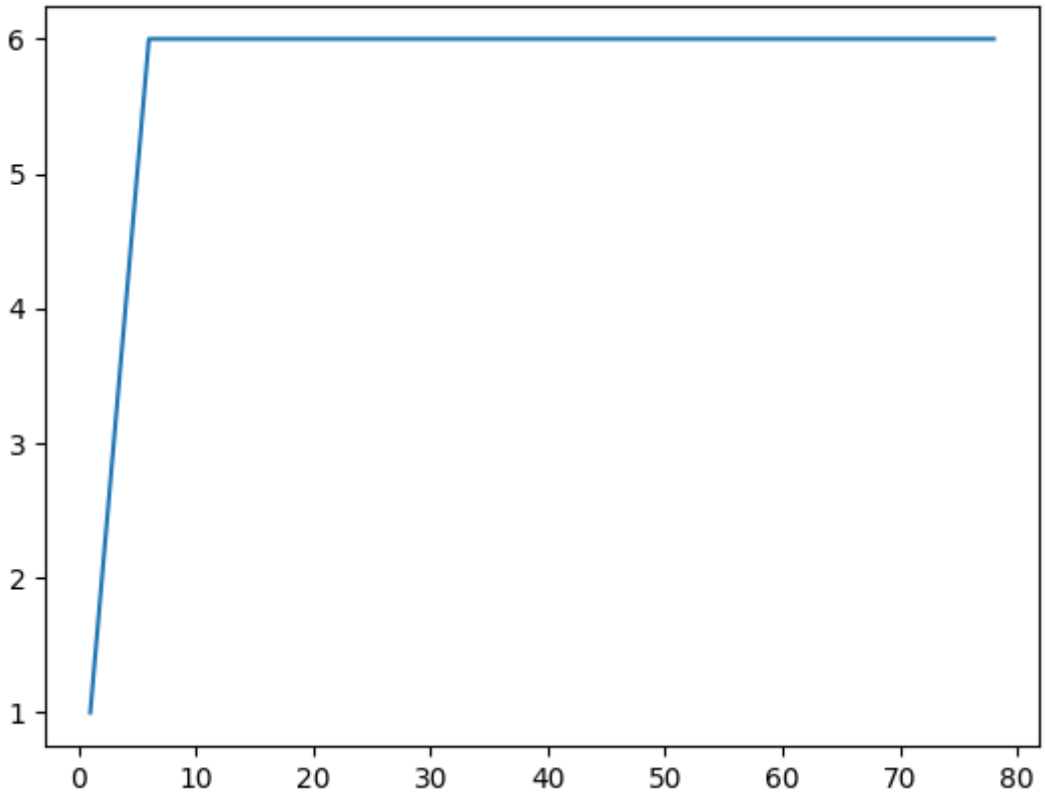
./program -N 6 -T 5 -S 40 -L 3 -U 6 -G 5 -F file



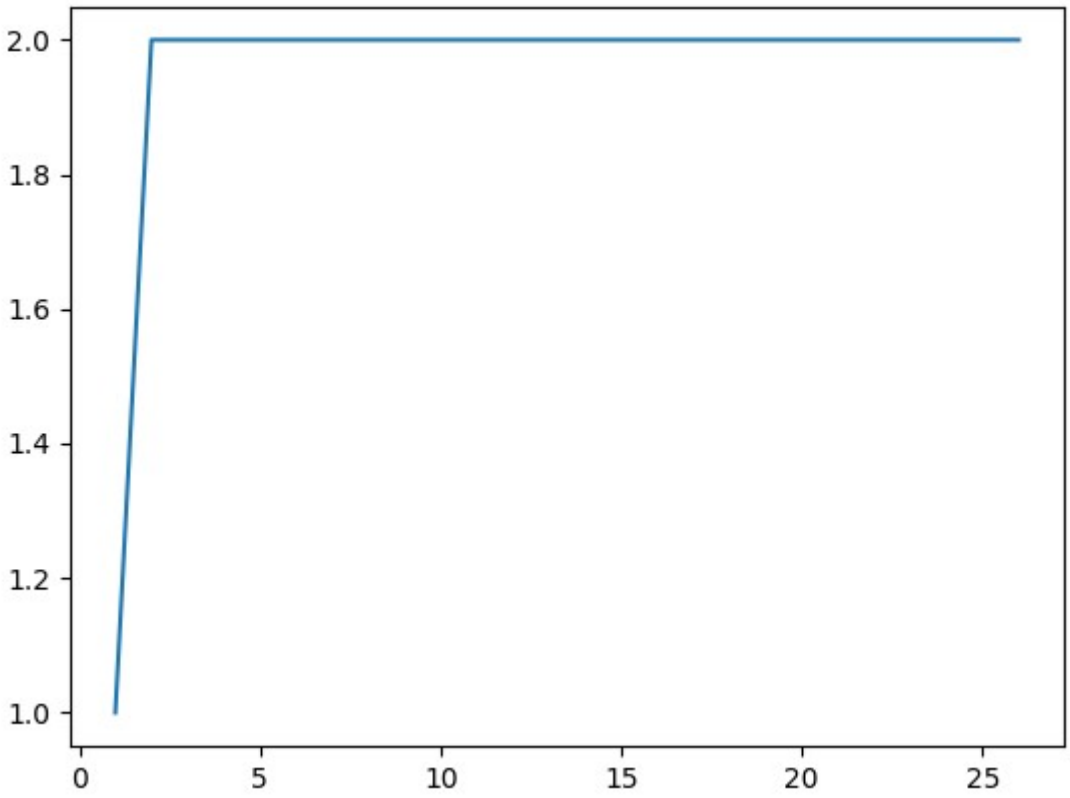
Plates at counter



Plates at kitchen

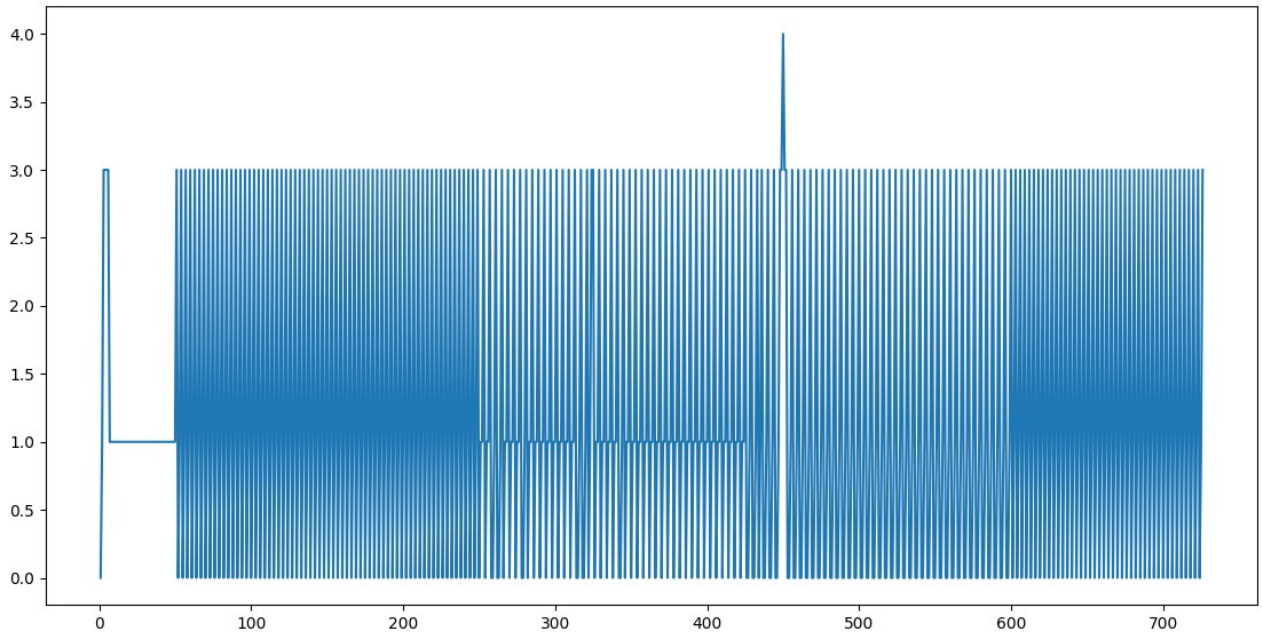


Students at counter

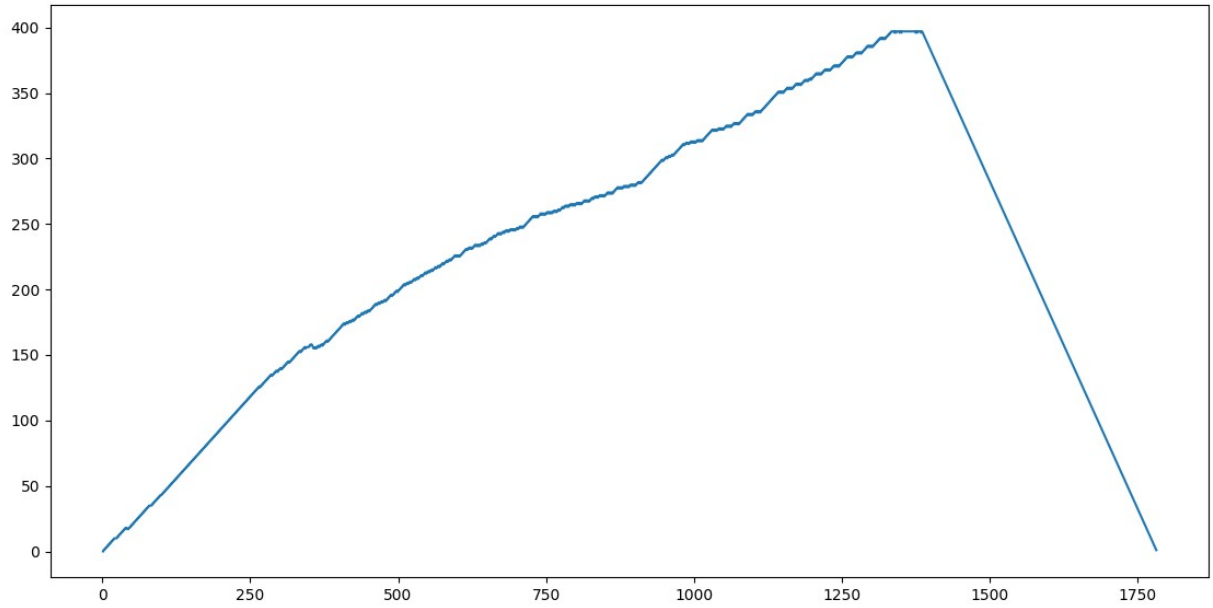


Graduated Student at Counter

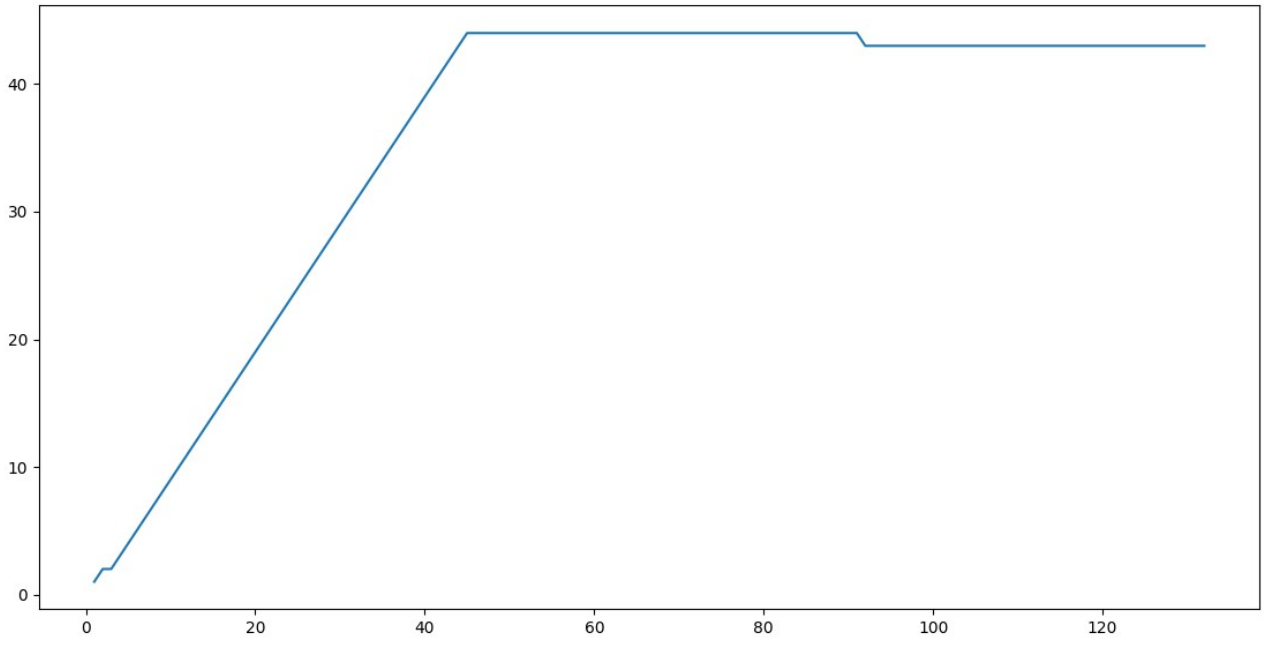
./program -N 44 -T 10 -S 44 -L 3 -U 44 -G 22 -F file



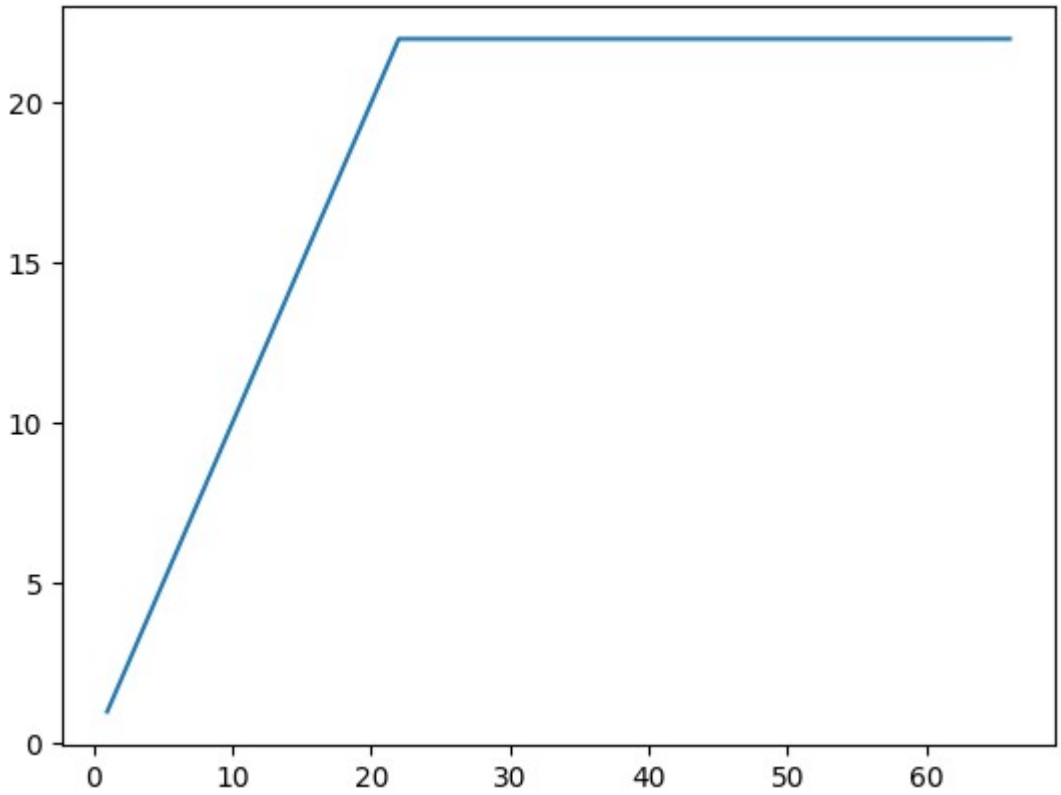
Plates at counter



Plates at kitchen



Students at counter



Graduated Student at Counter
When I try to run with 10000 processes(N+U+G), the program works successfully.
Note : Grad. and student plot ends sometimes with max number that means each students and grad. attend last round each time orderly(but there is no queue semophore determine it. They finished last round.

→ Compiler and Run :

Makefiles Commands:

```
make  #that command compiler
make clean #that command cleans all object files
```

To run :

```
./program -N 3 -T 5 -S 4 -L 13 -U 8 -G 2 -F filePath
```

→ Note : 1) I did bonus part. Please write argument like above command.
2) Student and Graduated Student console print differents. In this way, you can easily see the graduate student.

Thanks :)