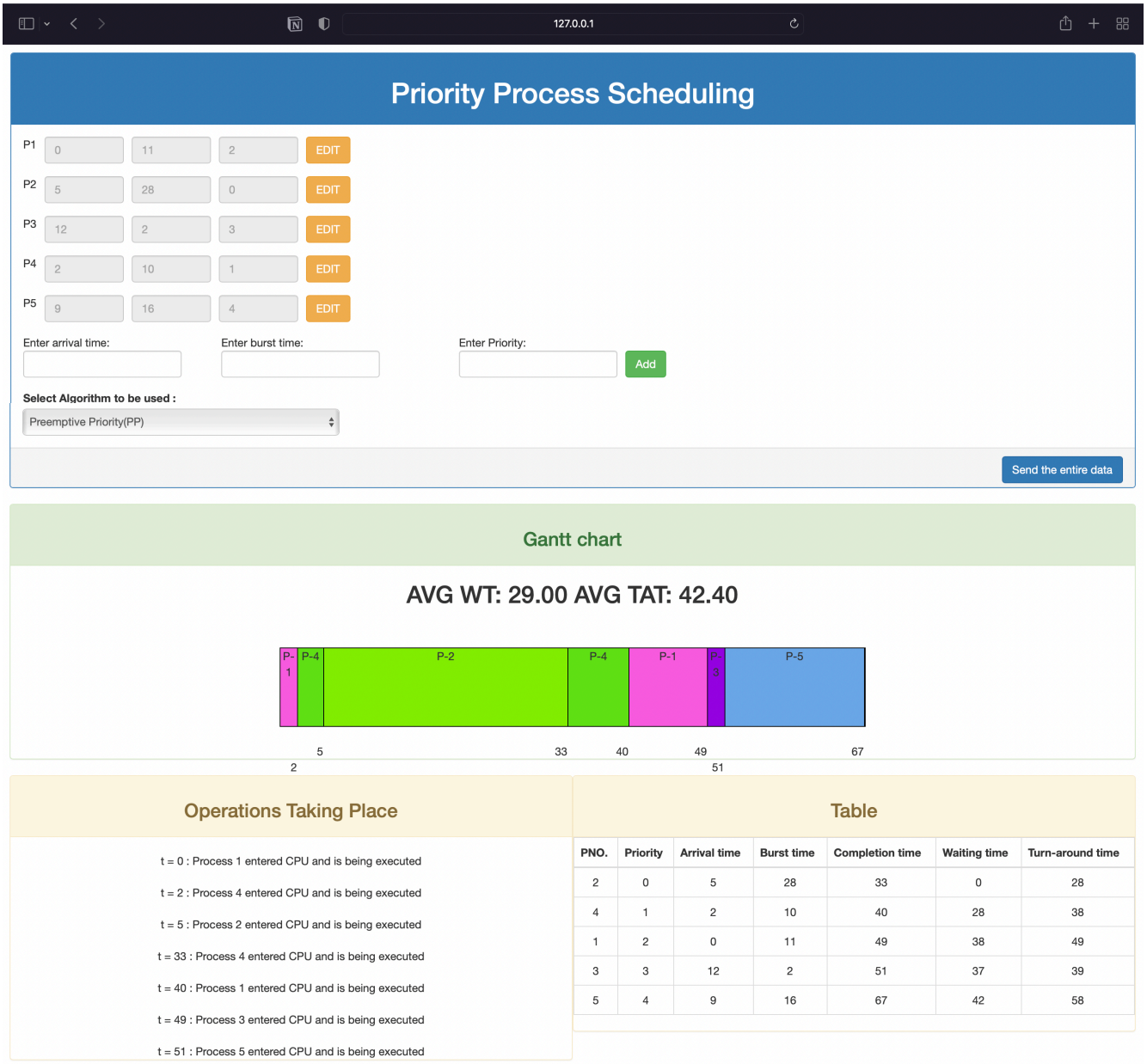
Process Scheduling

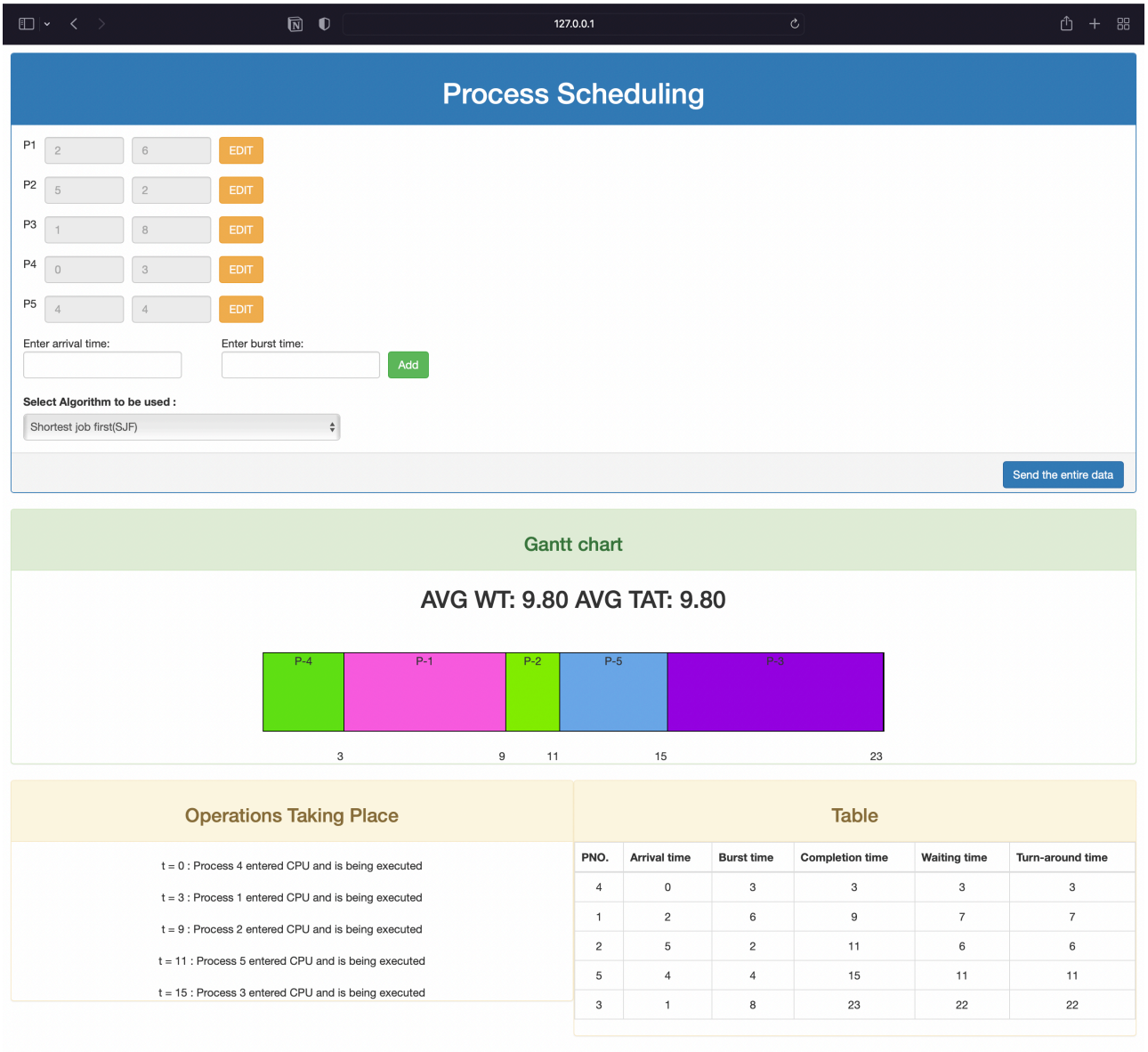
Schedulers that are implemented successfully:

1. Priority Scheduling



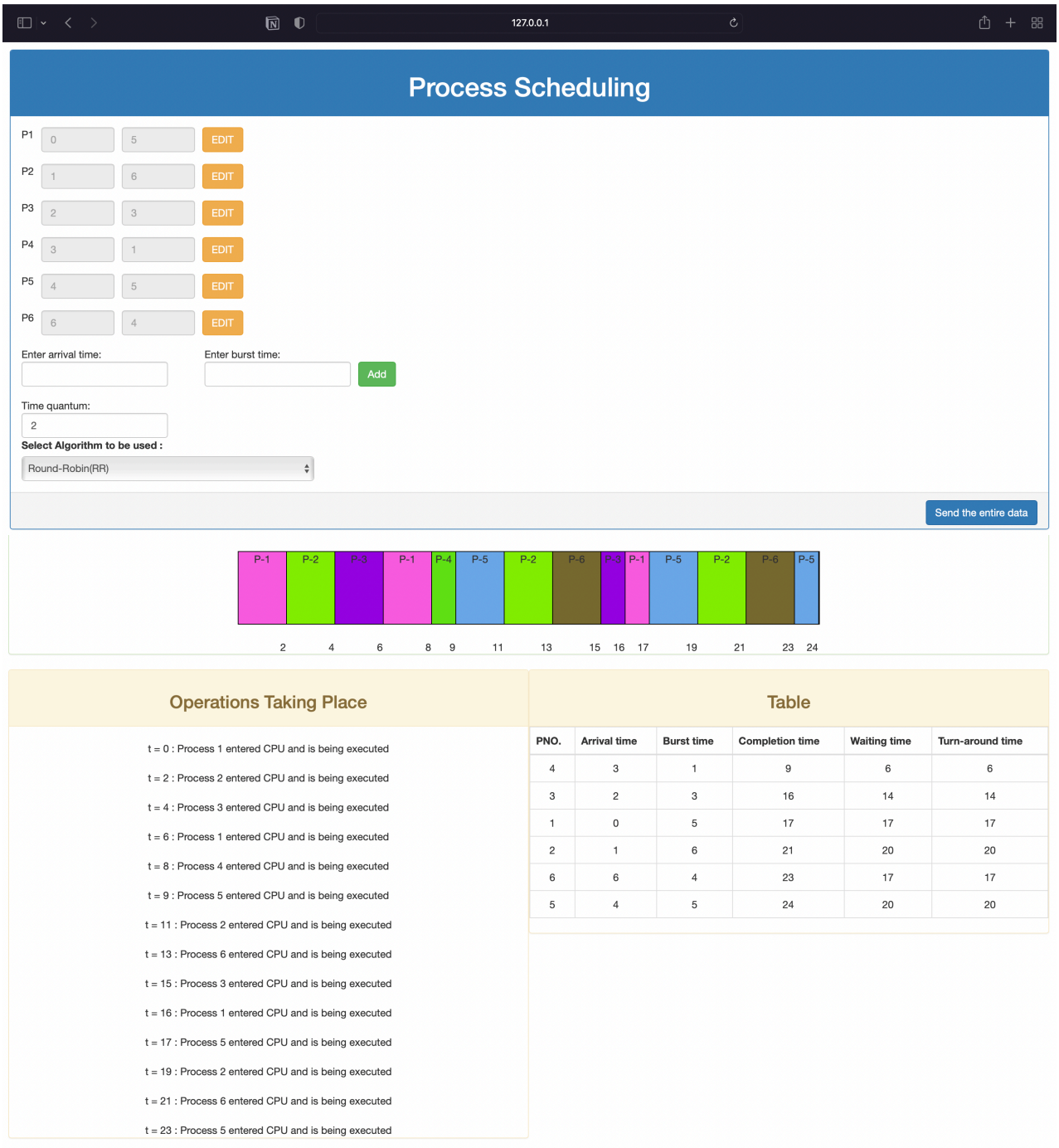
PNO.	Priority	Arrival time	Burst time	Completion time	Waiting time	Turn-around time
2	0	5	28	33	0	28
4	1	2	10	40	28	38
1	2	0	11	49	38	49
3	3	12	2	51	37	39
5	4	9	16	67	42	58

2. Shortest Job First Scheduling (Non-preemptive)



PNO.	Arrival time	Burst time	Completion time	Waiting time	Turn-around time
4	0	3	3	3	3
1	2	6	9	7	7
2	5	2	11	6	6
5	4	4	15	11	11
3	1	8	23	22	22

3. Round-Robin Scheduling



Operations Taking Place

t = 0 : Process 1 entered CPU and is being executed

t = 2 : Process 2 entered CPU and is being executed

t = 4 : Process 3 entered CPU and is being executed

t = 6 : Process 1 entered CPU and is being executed

t = 8 : Process 4 entered CPU and is being executed

t = 9 : Process 5 entered CPU and is being executed

t = 11 : Process 2 entered CPU and is being executed

t = 13 : Process 6 entered CPU and is being executed

t = 15 : Process 3 entered CPU and is being executed

t = 16 : Process 1 entered CPU and is being executed

t = 17 : Process 5 entered CPU and is being executed

t = 19 : Process 2 entered CPU and is being executed

t = 21 : Process 6 entered CPU and is being executed

t = 23 : Process 5 entered CPU and is being executed

Table

PNO.	Arrival time	Burst time	Completion time	Waiting time	Turn-around time
4	3	1	9	6	6
3	2	3	16	14	14
1	0	5	17	17	17
2	1	6	21	20	20
6	6	4	23	17	17
5	4	5	24	20	20

CMSC312-OS-Simulator / ossim / mat / views.py

Memory Management

1. Main memory

Every process that has to execute on the CPU needs a certain amount of memory. Memory management deals with the allocation of requested memory to each process for its execution. It is required in an operating system because the size of the main memory is limited, but the number of process that a computer can execute is not. Therefore, OS should make sure memory is allocated efficiently, and memory wastage is minimum.

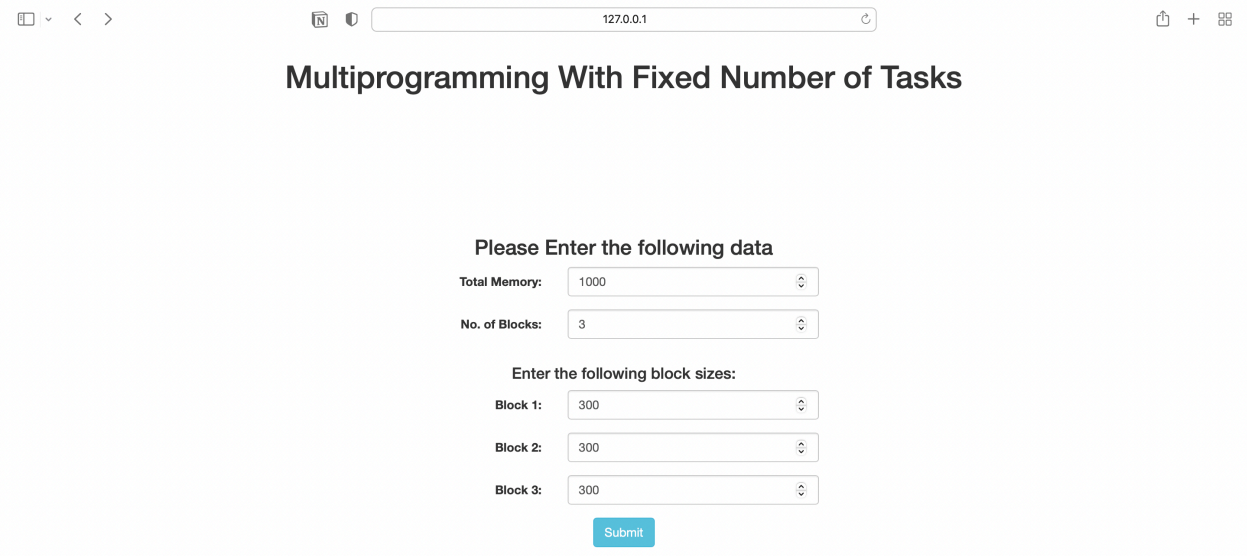
As a process enter the system, they are put into an input queue. The OS takes into account the memory requirements of each processes and the amount of available memory. When a process is allocated space, it is loaded into memory and it can then compete for CPU time. When a process terminates, it releases its memory, which the OS may then fill with another process from the input queue.

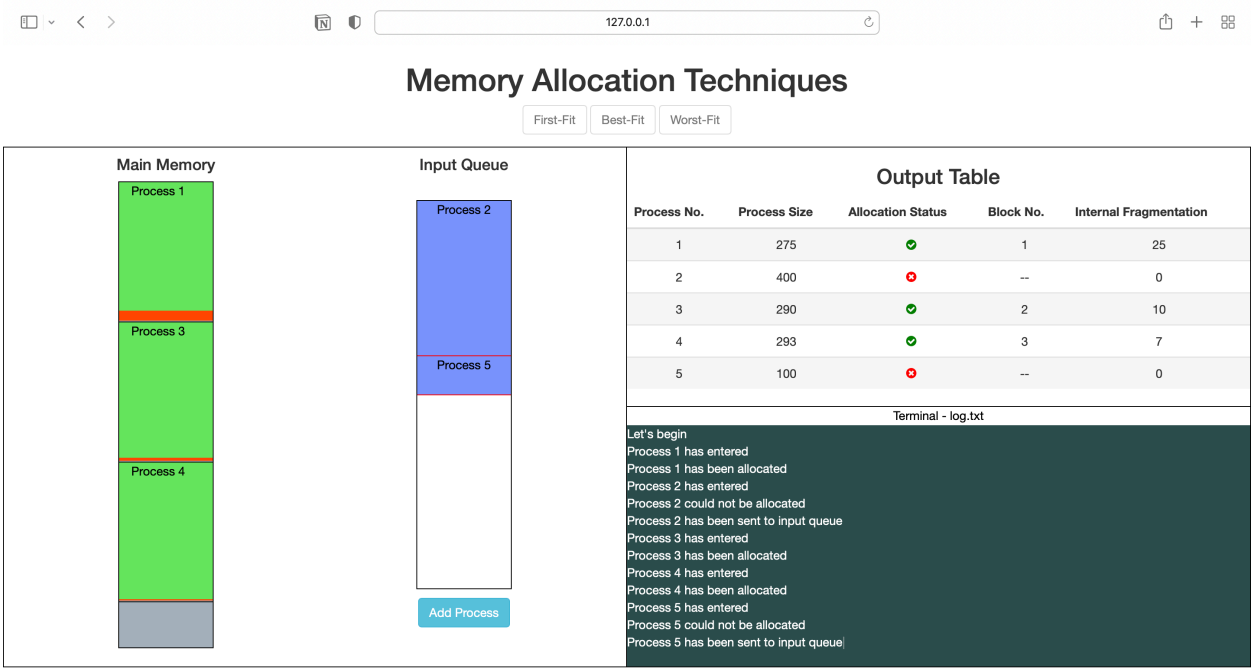
According to Chapter 8 - Main memory, main memory is thought of a continuous array of locations. There are two ways we can utilize this continuous memory to accommodate for processes are fixed- size partition or MFT and variable size partition or MVT. Each partition has their own First Fit, Best Fit and Worst Fit.
Please visit Wiki page in the “Memory Allocation” to read more about the two partitions’ advantages and disadvantages

Users are able to choose Main Memory size, Input Queue Size, and the kind of Fit they want.

Below is the some examples how MFT and MVT of this project work

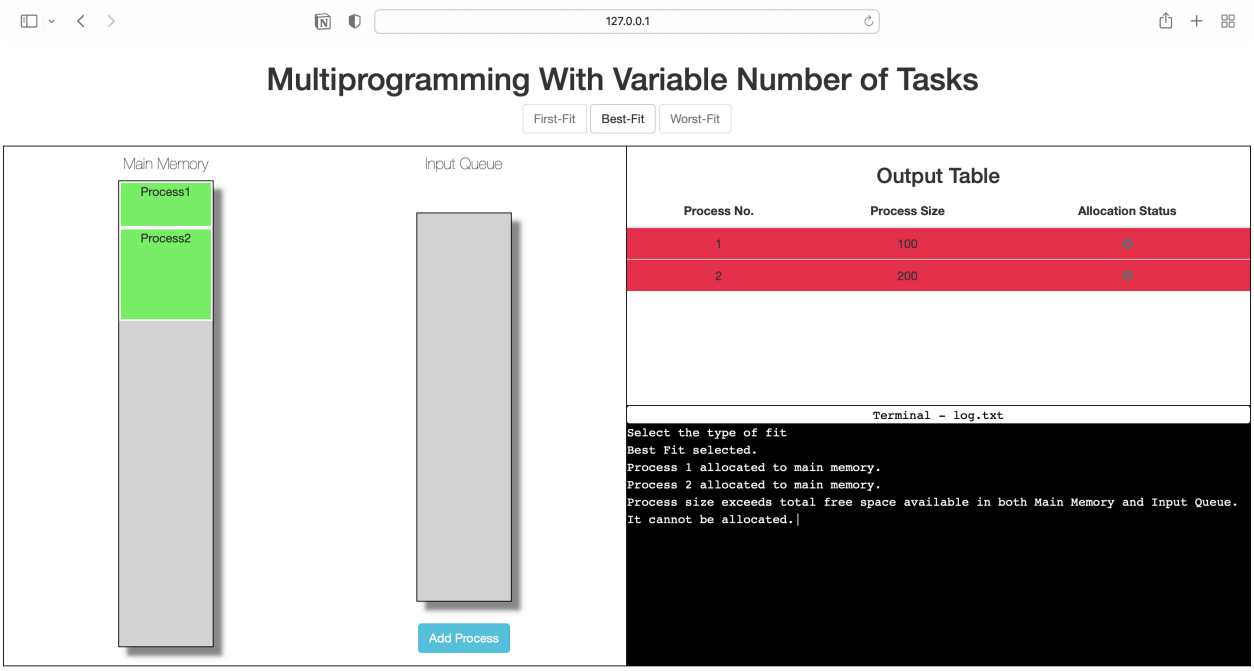
MFT



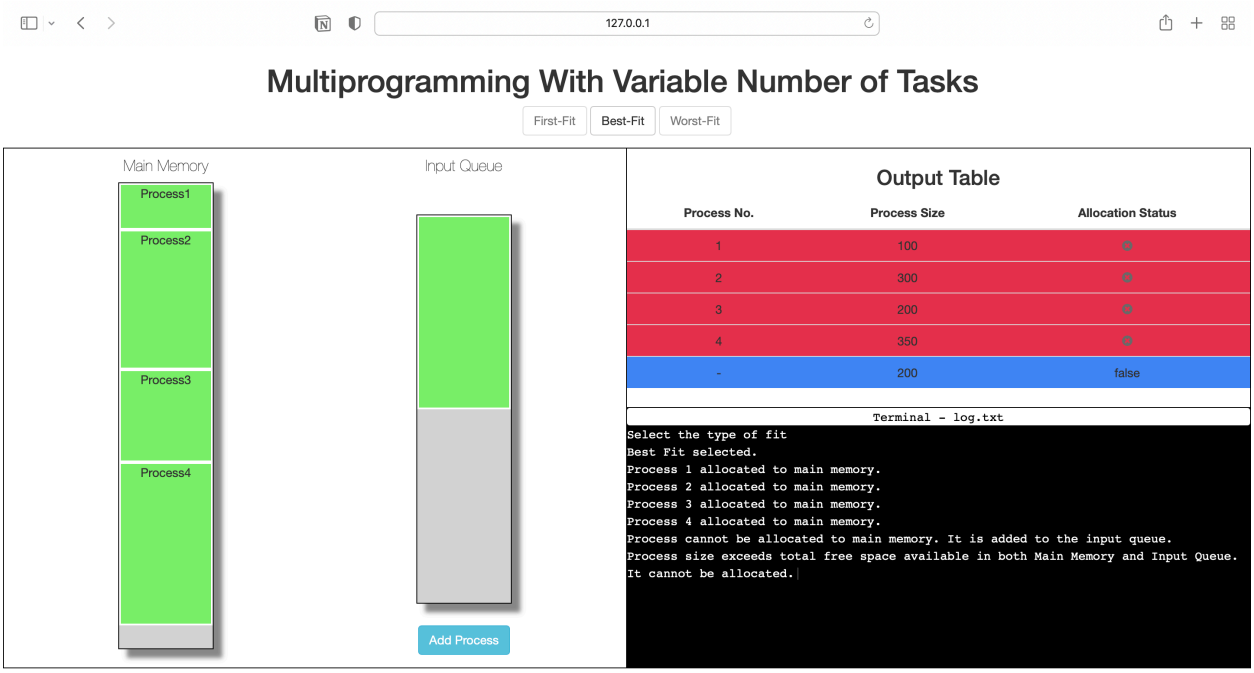


MVT

This is an example when **Total Main Memory** is 1000 and **Input queue size** is 2 but **process 3** is 800. Process 3 size exceeds total free space available in both Main Memory and Input Queue so it is not added to Input Queue. *(Ignore Allocation Status)**



This is an example when the **Total Main Memory** is 1000, **Input queue size** is 400 so when main memory is full but the process size doesn't exceed input queue size, it will be added to input queue. Otherwise, it cannot be allocated.



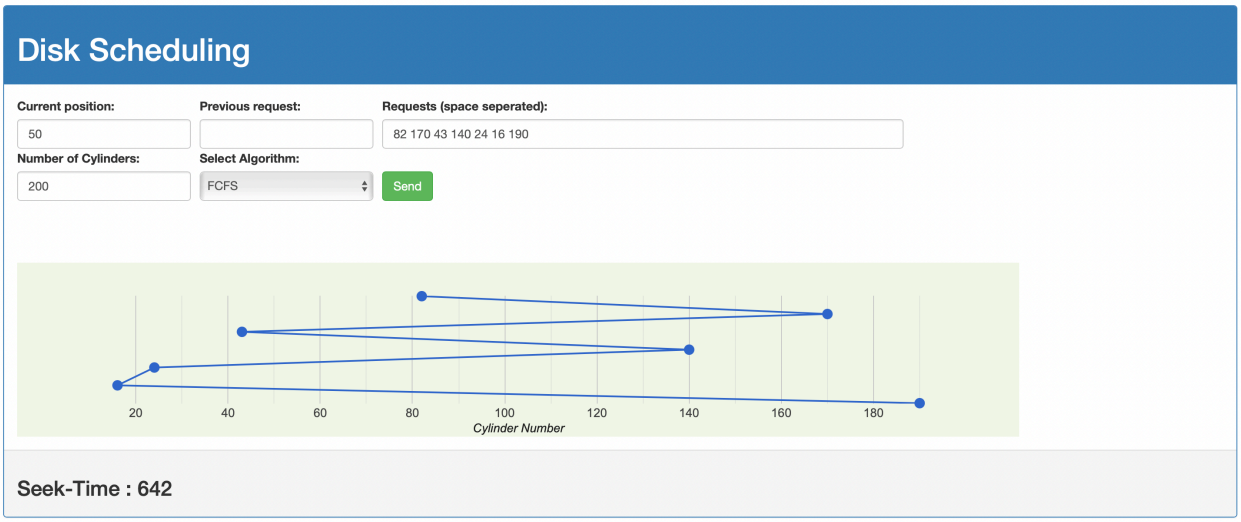
I/O Management

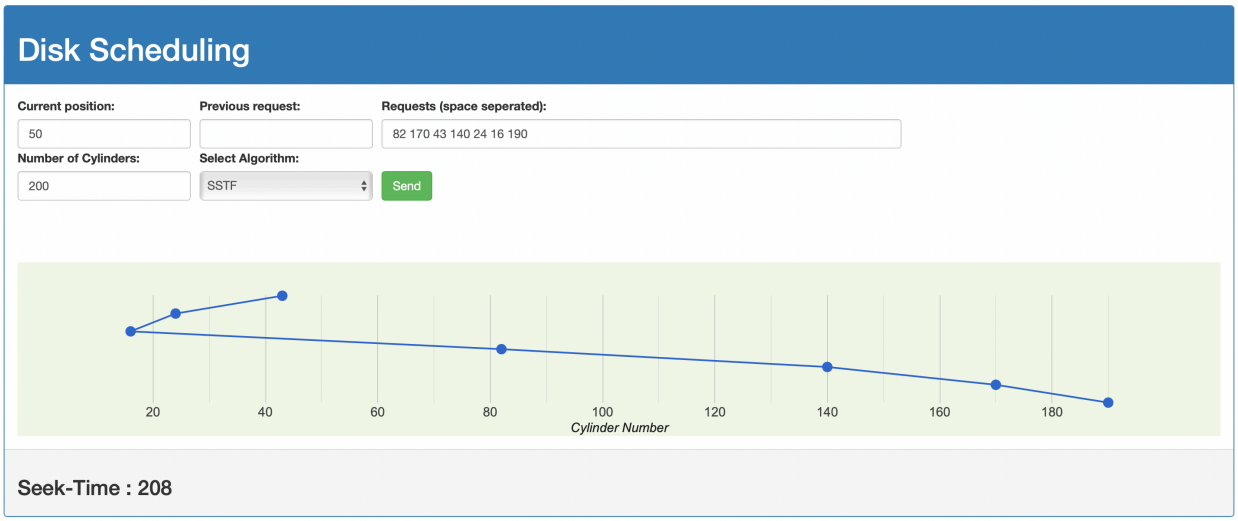
CMSC312-OS-Simulator / ossim / disk / utils.py

Disk scheduling is done by operating system to schedule I/O requests arriving for the disk so disk scheduling is also known as I/O scheduling. I/O scheduling is important because multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Therefore, other I/O requests need to wait in the waiting queue and need to be scheduled.

The file system may reside entirely on a single disk, or it may spread out on a number of disks. Therefore, all processes that ask for disk I/O services are competing for access to the same physical disk or set of physical disks. All processes must do at least one burst of disk I/O initially load a program to run. Any given disk can perform only on access at a time. Thus if there are several processes requesting for an access, the operating system must establish some order for the service of requests.

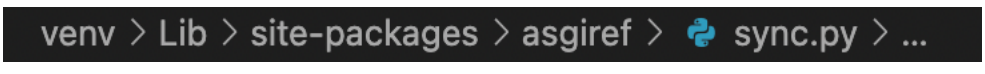
I/O requests are managed by Devices Drivers in collaboration with some system program inside the I/O device. The request are served by OS using three simple segments: I/O traffic controller, I/O scheduler and I/O device handler. I/O device handler manages the I/O interrupts and scheduling algorithms. I/O handling algorithms are FCFS, SSTF, SCAN, LOOK, C-SCAN, and C-LOOK. I choose to implement SSTF and FCFS. SSTF is common and has a natural appeal because it increases performance over FCFS.





Multi-threading and multi-CPU

Threading is just one of many ways concurrent programs can be built. I found there are several strategies in building concurrent program in Python and each of them is suitable in different scenarios. I personally think it is challenging to use Python for multithreading work. There is a thing called global interpreter lock (GIL) which prevents multiple threads of Python code from running simultaneously. Different from other language such as C++ or Java, Python multithreading module doesn't quite behave the way we would expect it. Below is the direction to check on the code. If you use VScode, you can do **Edit -> Find in files -> search for "Thread" or "threading"**), that way is easier to find the files that need to be checked.



GIT Repository

https://github.com/Huyenhuynh81/CMSC312-OS-Simulator.git

How to run the Operating System Simulator

You need to create a virtual environment to run this operating system simulator
If you have not install python then you need to download latest version of the language.
This is a guideline how to run the operating system on Mac

Step 1: Go to terminal. Check if the latest version of the language is install: `python3 —version`
Step 2: Install pip: `sudo easy_install pip`
Step 3: Install virtualenv for python: `sudo pip install virtualenv`
Step 4: Create a virtual environment:
- Navigate to Desktop
- Type: virtualenv <choose a name for your virtualenv> (Ex: Doraemon)

Step 5: Move the project folder (CMSC312-OS-Simulator) to the virtualenv you just created
Step 6: In terminal, navigate to Doraemon
Step 7: Activate the environment : `source bin/activate`
- To check if you are in the virtual environment, you will see `(Doraemon)` at the beginning

Step 8: Install the latest official version of Django: `sudo pip install django==3.0.1`
Step 9: Navigate to OS-Simulator/ossim
Step 10: Start the simulator: `python manage.py runserver`
Step 11: Copy the server address **`http://127.0.0.1:8000`** and run it in a browser

```
System check identified 10 issues (0 silenced).

You have 5 unapplied migration(s). Your project may not work properly until you apply
the migrations for app(s): admin, auth.
Run 'python manage.py migrate' to apply them.
October 31, 2021 - 13:40:07
Django version 3.2.8, using settings 'ossim.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[31/Oct/2021 13:40:23] "GET / HTTP/1.1" 200 5560
[31/Oct/2021 13:40:23] "GET /static/ossim/css/pluton.css HTTP/1.1" 200 3741
[31/Oct/2021 13:40:23] "GET /static/ossim/css/jquery.bxslider.css HTTP/1.1" 200 3892
[31/Oct/2021 13:40:23] "GET /static/ossim/css/style.css HTTP/1.1" 200 16433
[31/Oct/2021 13:40:23] "GET /static/ossim/css/bootstrap-responsive.css HTTP/1.1" 200 2
3211
```