How to Run these?

Run this> testit number_of_processes choice number

Choice Number:

1> Test case having I/O bound processes

2> Test case having CPU bound processes

3> Test case having default benchmark file

4> Test case having mixed processes

5> Test case having optimized process which uses MLFQ

For getting graph we need to add PLOT=YES in the running statement.

> make qemu SCHEDULER=<MLFQ,PBS,RR,FCFS> PLOT=YES > output.txt

Then run

> python3 graph.py

You will be able to see the required graph.

NOTE: please avoid keeping print statement in case of running testcase for generating graph as this may disturb the designed python splitting code. I have made sure that most of the corner cases will be taken care in this.

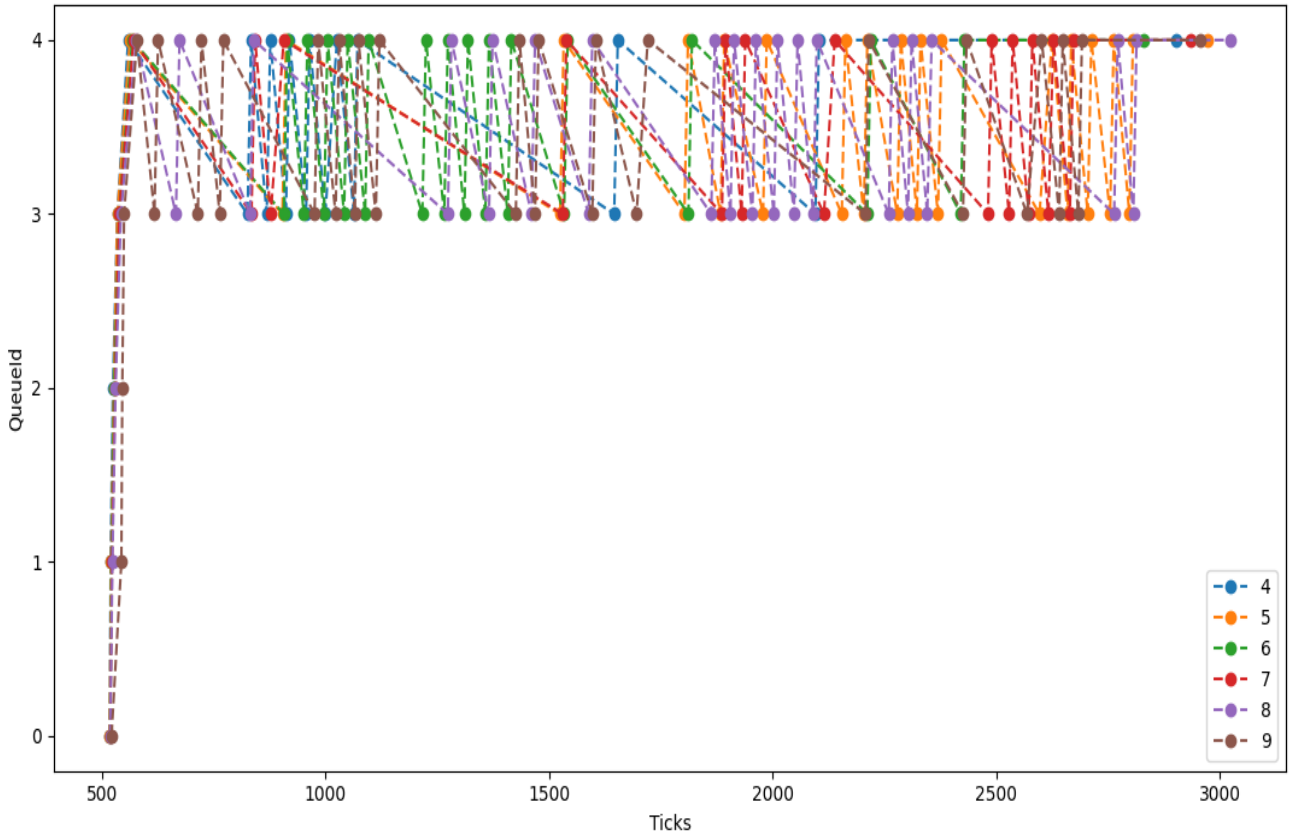
I implemented python code in graph.py and used appropriate files.

MLFQ PLOTS:

CPU Bound Processes:

How to execute:

Run > testit number_of_processes 2

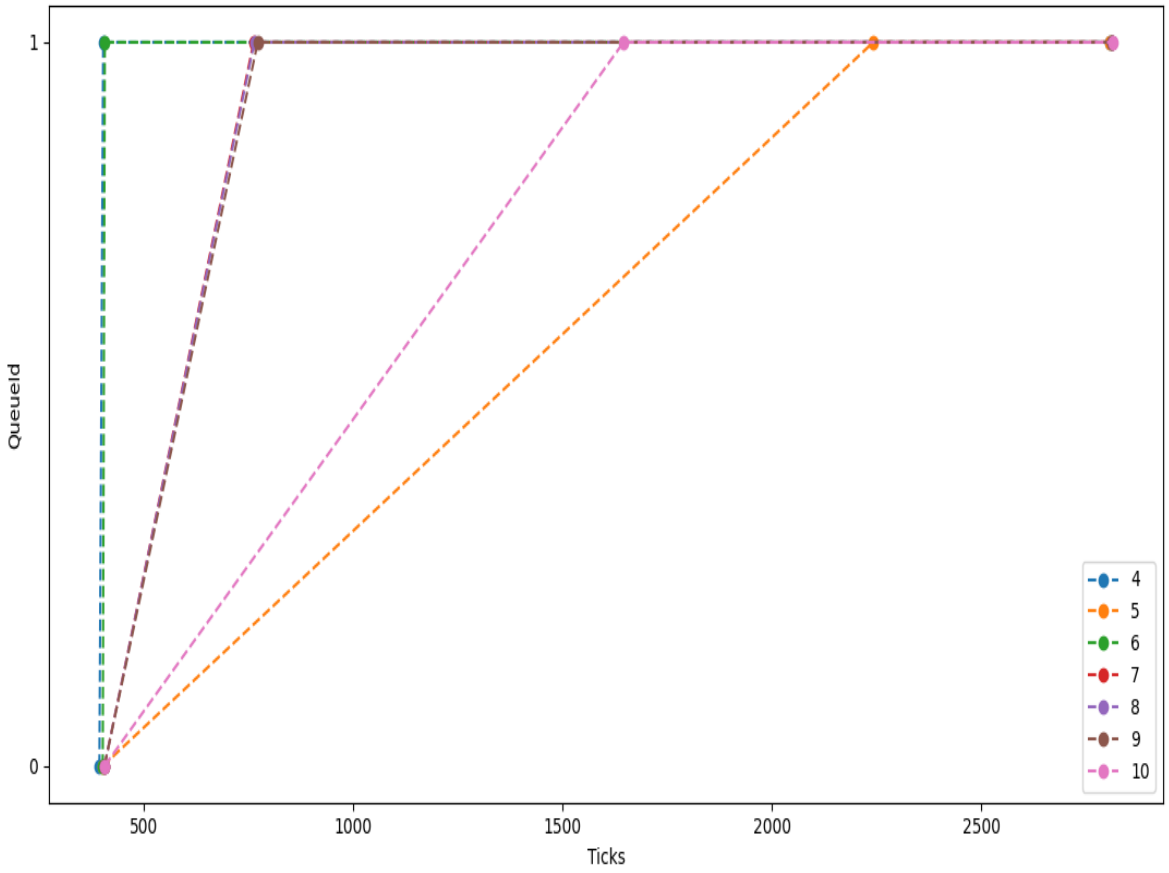


This is executed by giving 6 processes and aging time of 35ticks. They will mostly be in 3 and 4 th queues because they will consume their time slices most of the times and they need to wait because almost all of them will be in ready queue. Because of which there will be aging. Hence most of the processes in the example are also in the queue 3 and 4.

I/O bound:

How to Run?

Run > testit number_of_processes 1

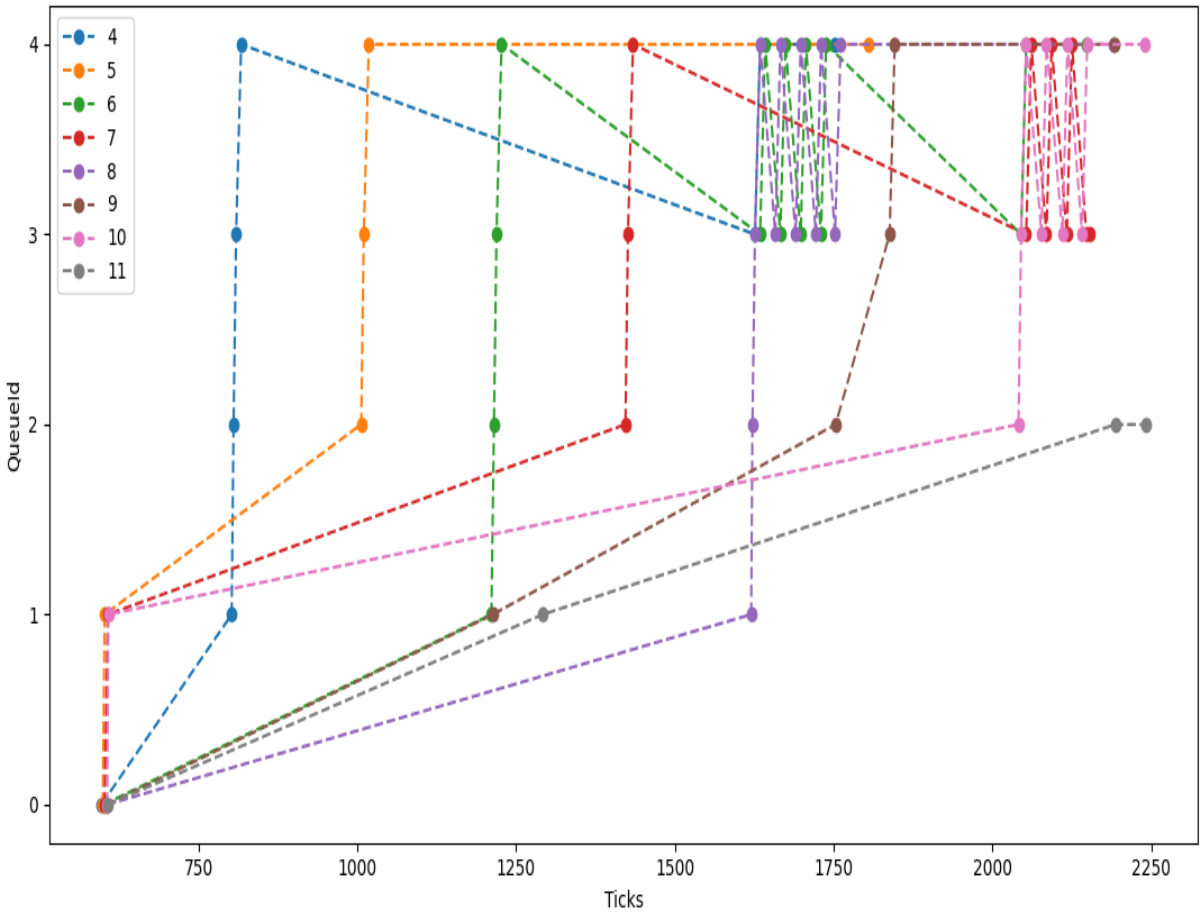


Here most of the processes are in queue 0 and 1 because these are the process which wont use CPU so they will complete their CPU requirement before their maximum allocated time is done. Hence, they will stay in their queue and almost in most of the times wont upgrade their queue because of aging because the time they spend in RUNNABLE state is less when number of processes are less.

Given Benchmark:

How to Run?

Run > testit number_of_processes 3

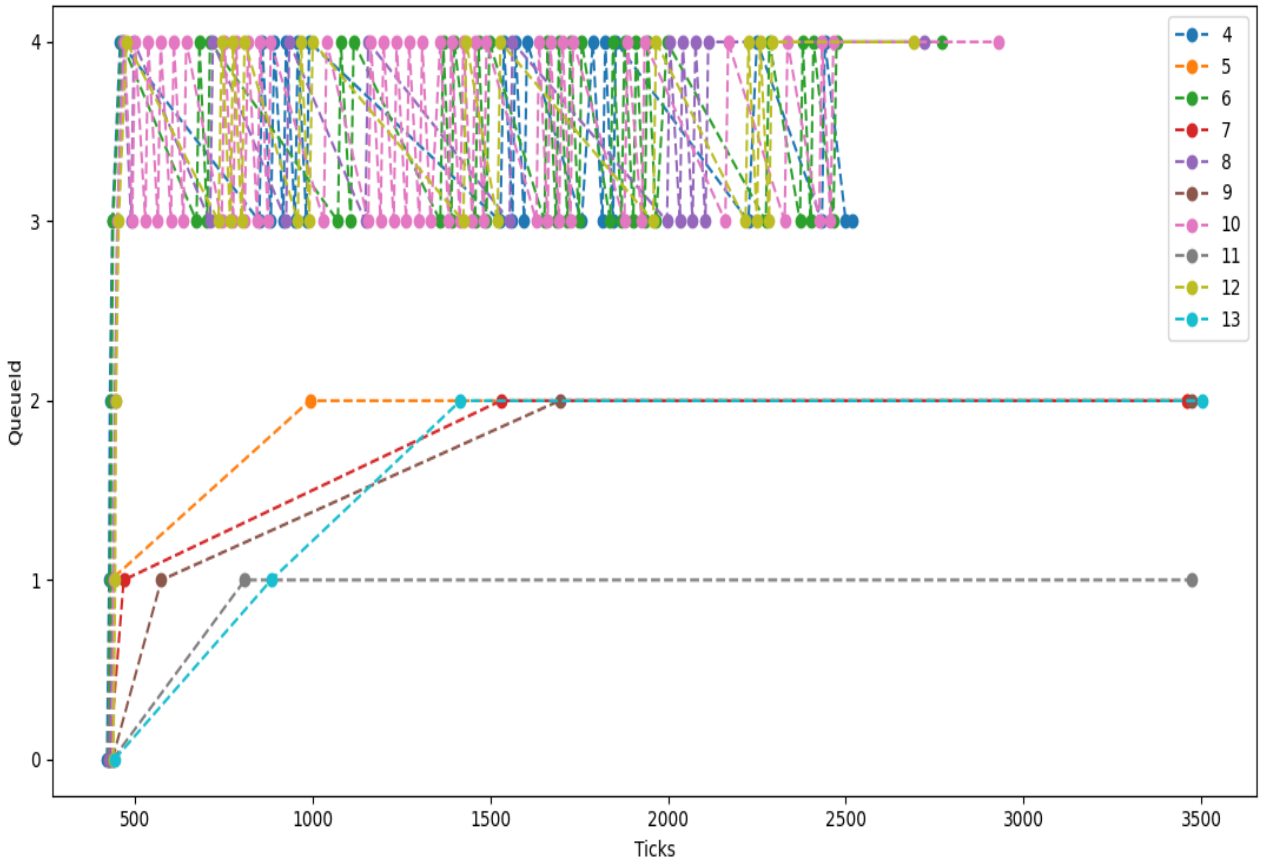


This is the graph for the given benchmark file given in moodle. This is approximately same as CPU + IO bound processes as evident from the graph. Note that there is aging and some are in 2nd queue and some shuffling between the 3rd and 4th queue which says that there is a good mix of usage of I/O and CPU device utilization.

Mixed (I/O and CPU bound Processes):

How to Run?

Run> testit number_of_processes 4

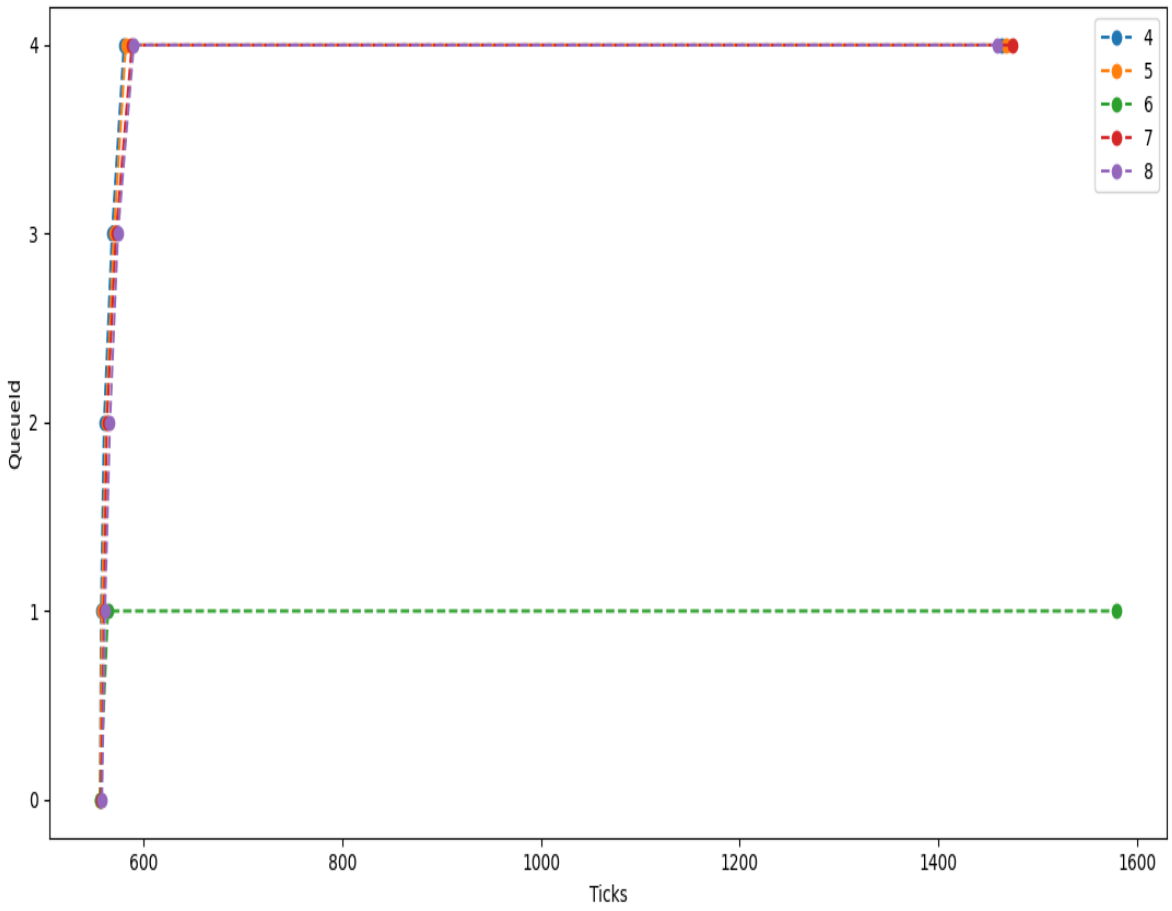


Observe that there are few processes which are staying in the lower queues and some are in the higher queues which are I/O and CPU bound processes.

Optimized Usage of MLFQ:

How to Run?

Run> testit number_of_processes 5



Here observe that the process with pid 6 is using the MLFQ very efficiently. The rest of the processes are CPU bound so they moved to higher queues.

How will Process Use MLFQ Efficiently:

It can use MLFQ efficiently if the process relinquishes the CPU before its CPU slice is getting completed. Because of this most of the time the process will stay in the highest queue and thus will be given highest priority when the scheduling algorithm ran. As seen above in the graph observe that process with PID 6 is executing in queue 1 most of the time because of its optimal usage of MLFQ.

Comparison factors need to be kept in mind:

Round Robin:

In RR we will be based on factors like it will preempt the process after fixed number of clock ticks if the process didn't relinquish the CPU before that. If it relinquishes then other process will be scheduled in circular fashion. Because of this nature both I/O and CPU bound processes will execute within some time frame and they can do their job depending on the device in case of I/O bound and the CPU bound will keep on switching the processes and will in turn support them as well. Although the I/O bound are not right away given highest priority they will served in some time frame.

Priority Based Scheduling:

This algorithm will run the process with highest priority. So this will give highest priority to I/O bound processes because of which they will be scheduled immediately and will go to I/O operations after that, unlike few other algorithms where it will schedule the CPU bound which asks I/O to wait uselessly. So this algorithms works just fine if there are I/O and CPU bound processes.

MLFQ:

This is again an algorithm which utilizes the priority of the processes very well. It will try to schedule the processes with the highest priority because of which the processes which require small number of ticks will be executed first and after that they can do their own job. Because of this type of algorithm the I/O bound processes which utilizes the least amount of CPU ticks will say at the top of the queues and they will new scheduling these processes first which increases the performance of the utilization of resources.

FCFS:

This is perhaps the most naïve way of doing things. This is because there can be a case where the I/O bound process which may be at the end of the queue to wait for CPU bound processes which will consume lot of time. Because of this kind of behavior, the I/O device will be kept empty this very poor utilization of the resources. This will work in the cases where the I/O are ahead of CPU bound

processes. So According to my analysis this is the worst algorithm if there are I/O bound and CPU bound processes.

The real performance will really depend on the specifics of that particular testcase hence we cannot say anything numerically. But below is my effort in trying to run a test case which consists of good mix of the I/O processes and CPU processes. Below is the detailed analysis:

Performance Comparison:

MLFQ:2629

RR: 2567

PBS:2689

FCFS: 3682

These are the values when I ran the benchmark program for the respective programs. Benchmark program consists of good combination of I/O and CPU processes.

In my comparison I found that Round Robin algorithm to be best. FCFS is found to be worst. This behavior may be because the CPU processes are being run and the I/O are waiting at the end and because of this although the I.O code takes little time it is not utilizing this property. We can say that MLFQ utilizes this property because it demotes the CPU processes, and it gives the I/O highest priority so this will also be faster than the FCFS algorithm. Coming to PBS we changed the priorities of the I/O and CPU bound processes in the code, we gave high priority to the I/O given this, it will also schedule the I/O bound process ahead of the CPU bound process hence utilizing those time slices efficiently. Hence the observed results. Among the top 3 it depends on many other factors like how many processes we run and many other factors.