OSN Assignment 5 : Modifying xv6

Report

T.H.Arjun
2019111012, CSD

Bonus - Plotting Graphs

For the bonus part of the assignment I made a test user program called load which is added to xv6 as a user program. Data is collected using print statements with a flag called PLOT (see instructions in README on how to plot) and wrote a python script named plot.py to plot the graphs using matplotlib. plot.py and load.c are modifiable according to needs of testing. For Plotting the ageing factor was set appropriately to show the aging.

The plot of Intelligent Process explains how a process could exploit MLFQ.

MLFQ PLOTS

Mixture of Processes

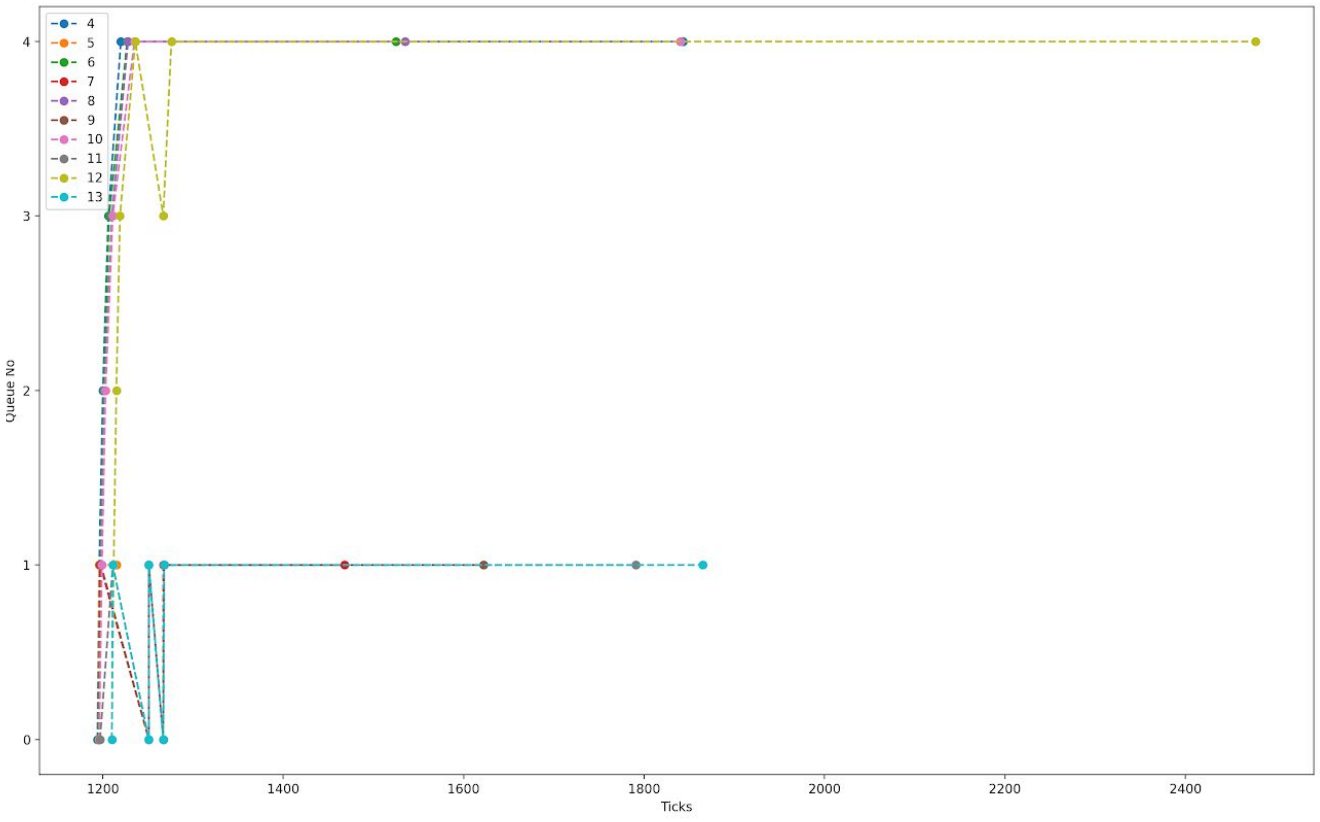


Fig 1. Mixture of IO Bound and CPU Bound Processes. Shows how IO Bound Processes with short CPU bursts don't utilise their full time slices in queue and remain at higher priority queues while CPU Bound processes move to higher queues because they don't voluntarily give up the CPU. The ageing phenomenon is also shown in this plot.

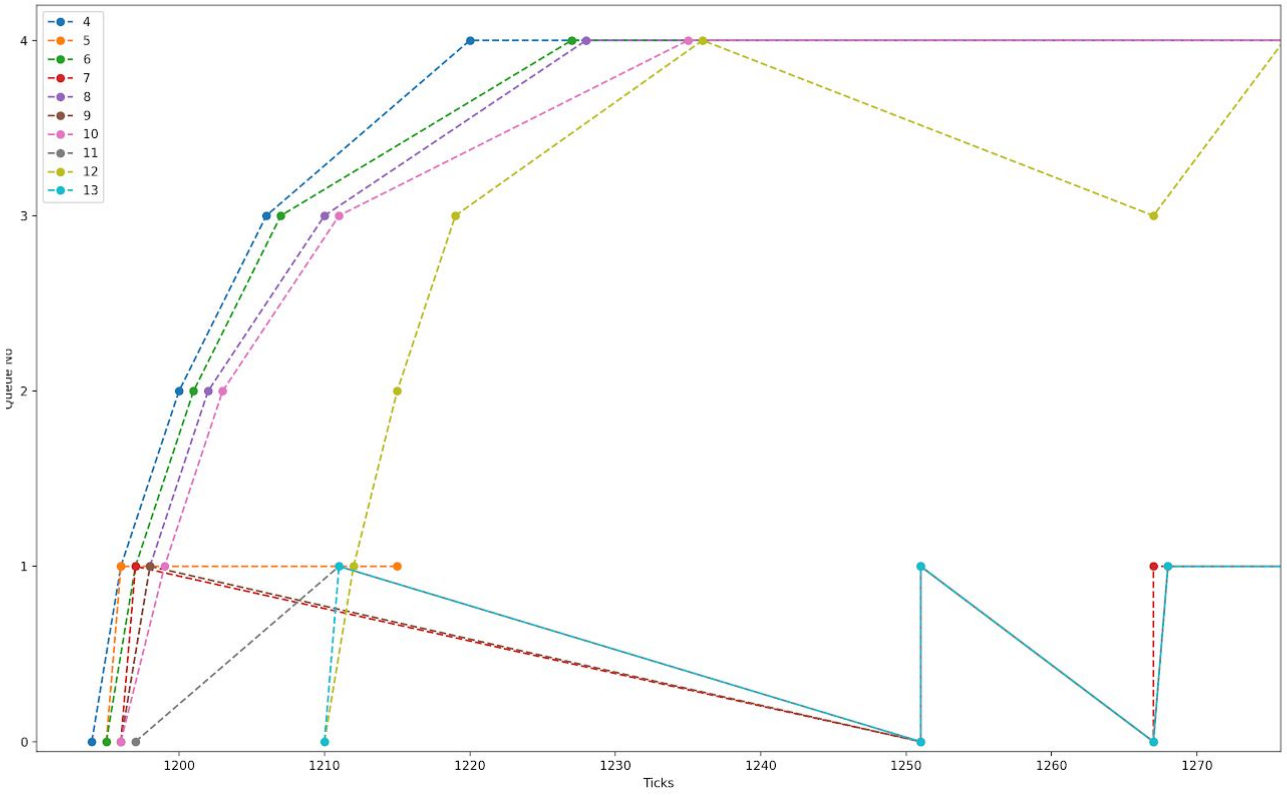


Fig 2. A Zoomed in version of the previous Plot, to show how each process behaves. The testcase used was load 5 10. It creates Even Number PIDs as CPU Bound Processes and Odd Numbered PIDs as IO Bound. This explains their positions in queues.

The Intelligent Process

Now we look at the test case of an intelligent process which exploits the scheduling algorithm by relinquishing the CPU before the time slice, and hence remaining in higher priority most of the time. For this test, the test case from load used was load 4 5. Which makes 5 processes out of which the first one is the intelligent one. Here Process with PID 4 is the intelligent one, the plot clearly shows how it is remaining in higher priority queues like 0,1 and at max moving to 2, while other processes move to the last queue 4. This is how a process can exploit the MLFQ Algorithm.

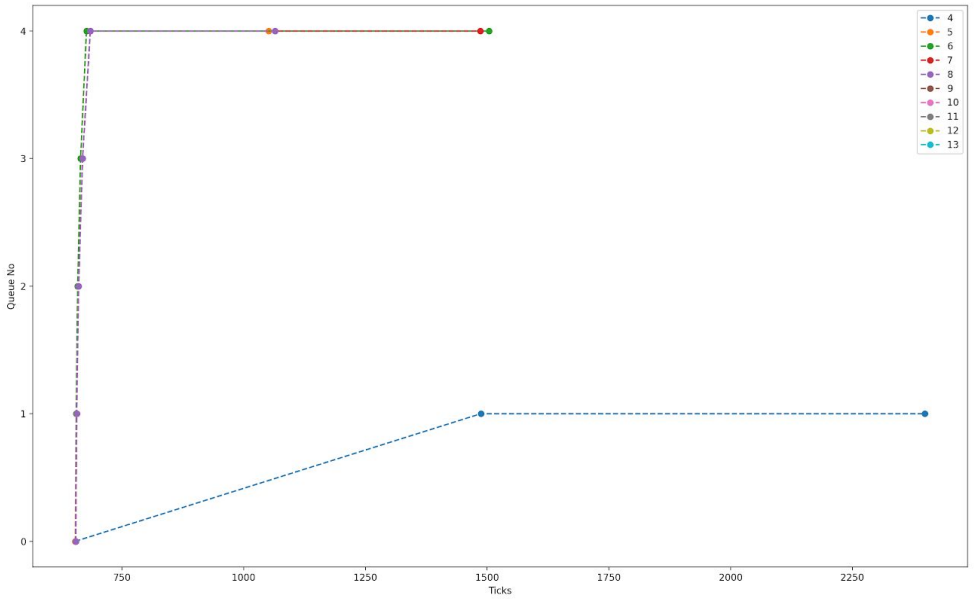


Fig.3 . The intelligent Process which exploits MLFQ (PID 4)

Given Benchmark Program

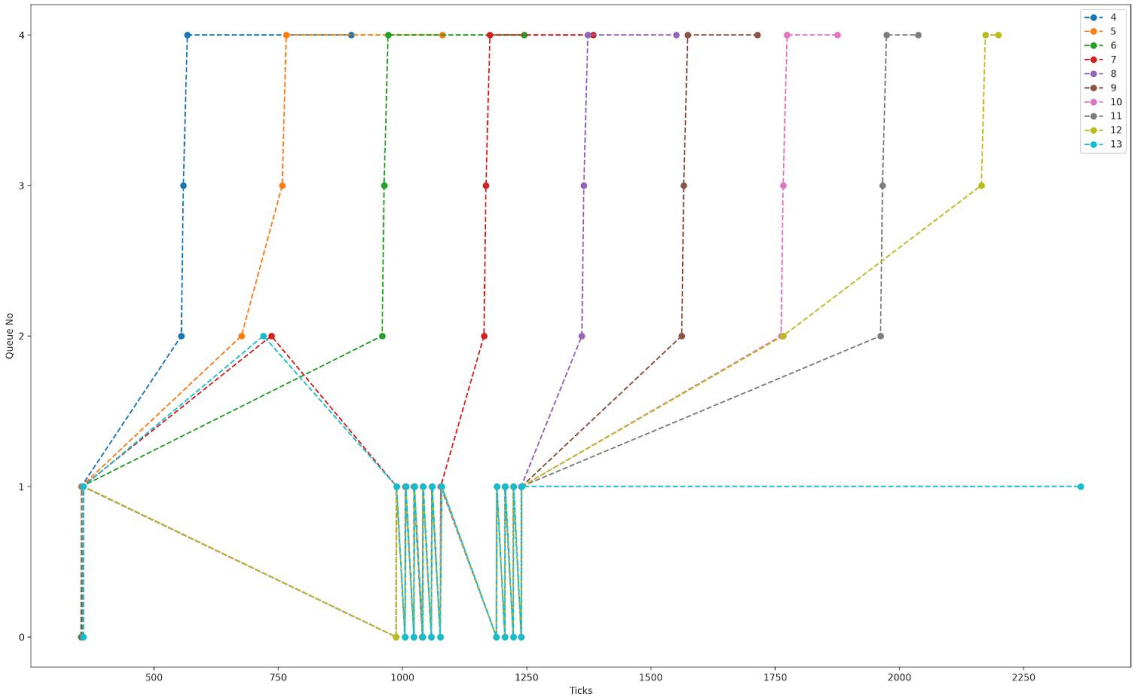
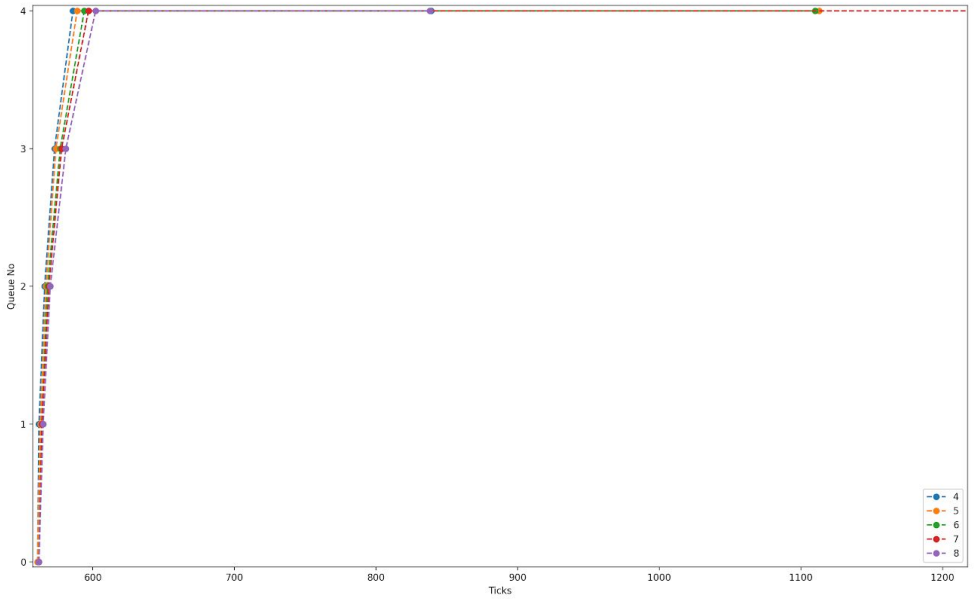


Fig 4. IO Bound remain in higher priority queues while CPU Bound moves to lower Priority queues.. (can be seen from PIDs)

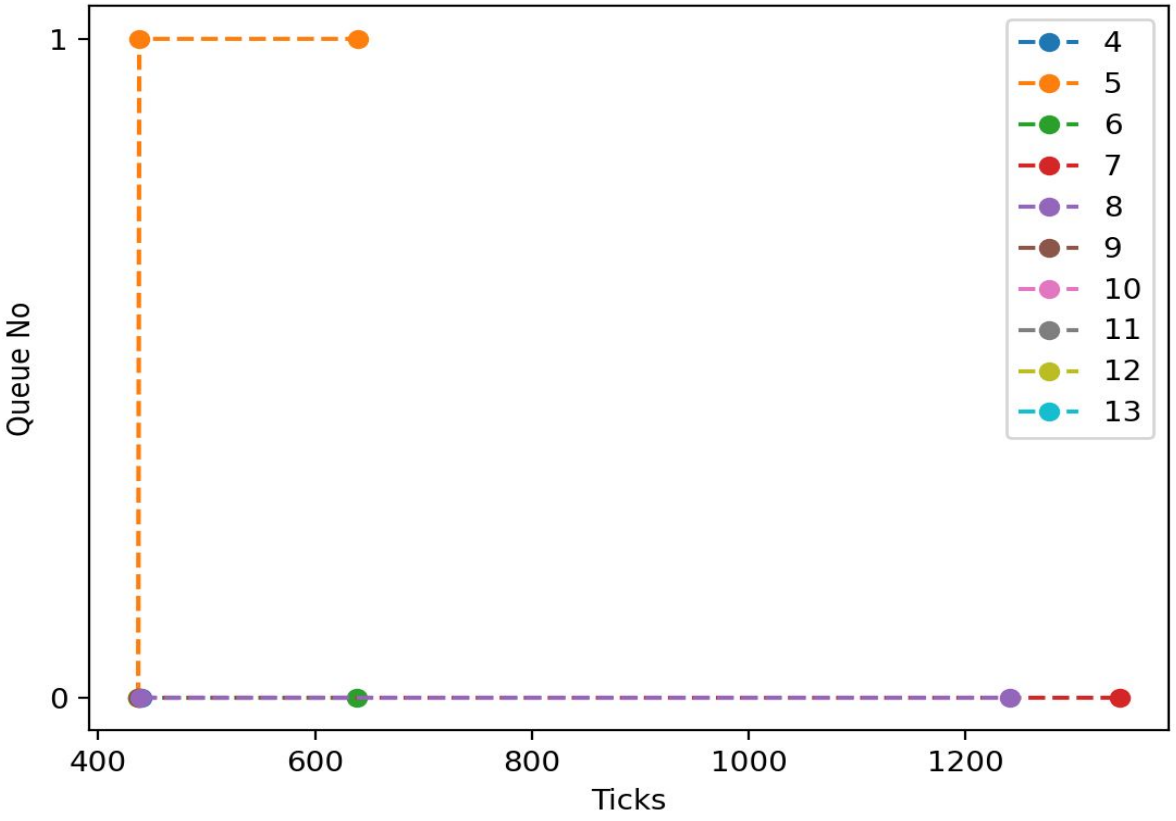
CPU Bound Process

CPU Bound Processes are CPU heavy and they don't relinquish CPU before their time slices and they move to lower priority queues like 3,4 due to the same reason.



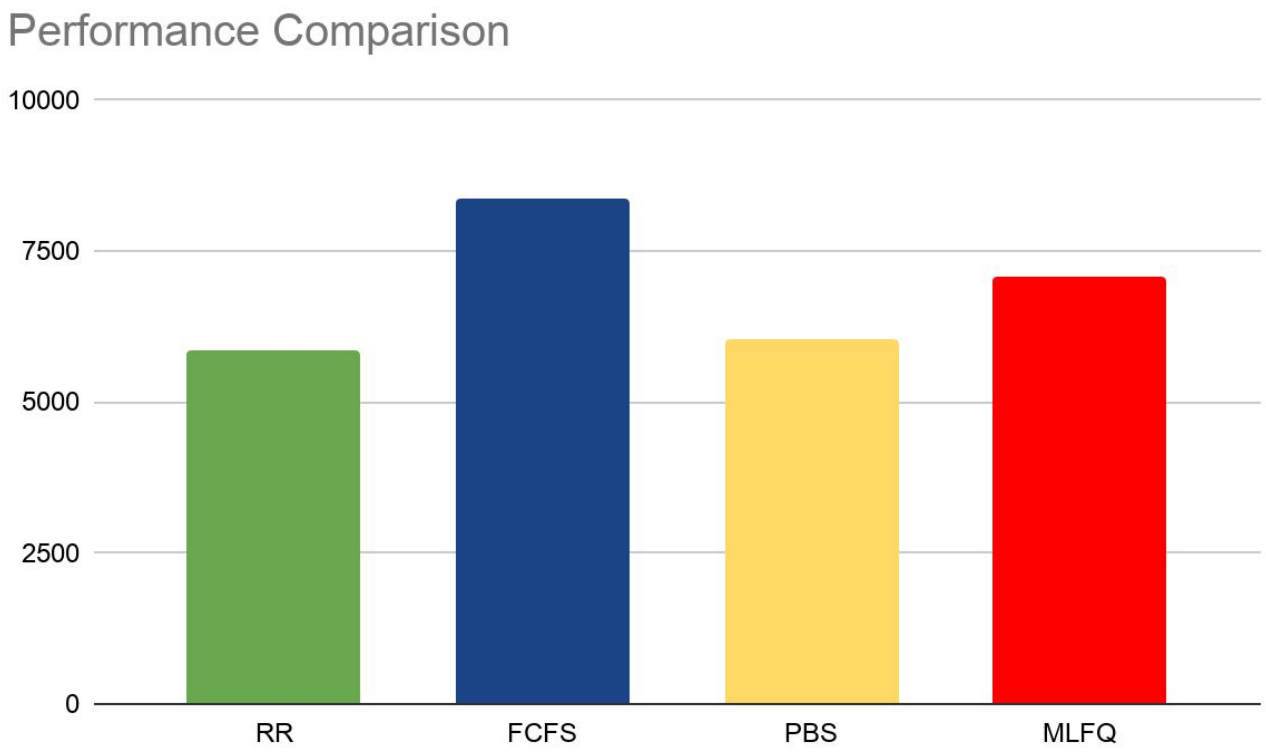
IO Bound Process

IO Bound Processes have shorter CPU Bursts and stays in Higher Priority Queues like 0, 1, 2 etc..



Performance Comparisons

I ran the Benchmark Program for 25 Processes and this is the result. Data collected is found in Data Folder. The Data is in terms of ticks that is end ticks and start ticks of PID 3 (benchmark program)



Ranking for the Given Benchmark:

1. RR
2. PBS
3. MLFQ
4. FCFS

Analysis:

Ticks for 25 Processes in Benchmark:

Algorithm Used	Ticks Consumed
RR	5840
FCFS	8354
PBS	5957
MLFQ	7086

Conclusion

The test included a good mix of IO Bound and CPU Bound Processes. In my Testing RR gives the best performance as at each tick a process is selected and executed so CPU Bound and IO Bound get the same Priority and get equal chances at the CPU if they are runnable. FCFS showed the worst performance as all CPU Bound Processes came first and the IO Bound Processes had to wait for them to get over, and there were no processes to run while the IO Bound functioned and were doing IO. PBS and MLFQ give similar results in this case since IO Bound Processes were given higher priority. If Priorities are assigned appropriately then PBS performs better than MLFQ. But it depends on priority allocation which depends on how the system call and user program are used.

RR

In RR we are yielding at each clock tick and hence each type of process (IO and CPU Bound) get equal chances. The IO Bound don't wait for CPU Bound even if they come first. Equal opportunity makes it faster.

FCFS

In FCFS, because there is no yield, if CPU Bound Processes come first then they take a lot of time and IO Bound have to wait for them to get over. And once IO Bound starts execution and goes to Sleeping there are no processes to run in their place with CPU hold thus taking a lot of time.

PBS

In PBS, the benchmark gave priority to IO Bound Processes, so they were executed whenever RUNNABLE. Hence it ran faster. This will depend on priority assignment as mentioned.

MLFQ

As shown in plots above the IO Bound Processes stay in higher priority queues and hence they are executed whenever possible. CPU bound used their slices and hence were pushed to lower priority queues and were chosen IO Bound were Sleeping hence the performance gain over FCFS.

Files Included:
plot.py - used for plotting (refer Readme)
Data Folder - has all the data used for plotting and comparisons
Graphs - has all the graphs included in the report

Finally a cool Plot for 25 Processes...