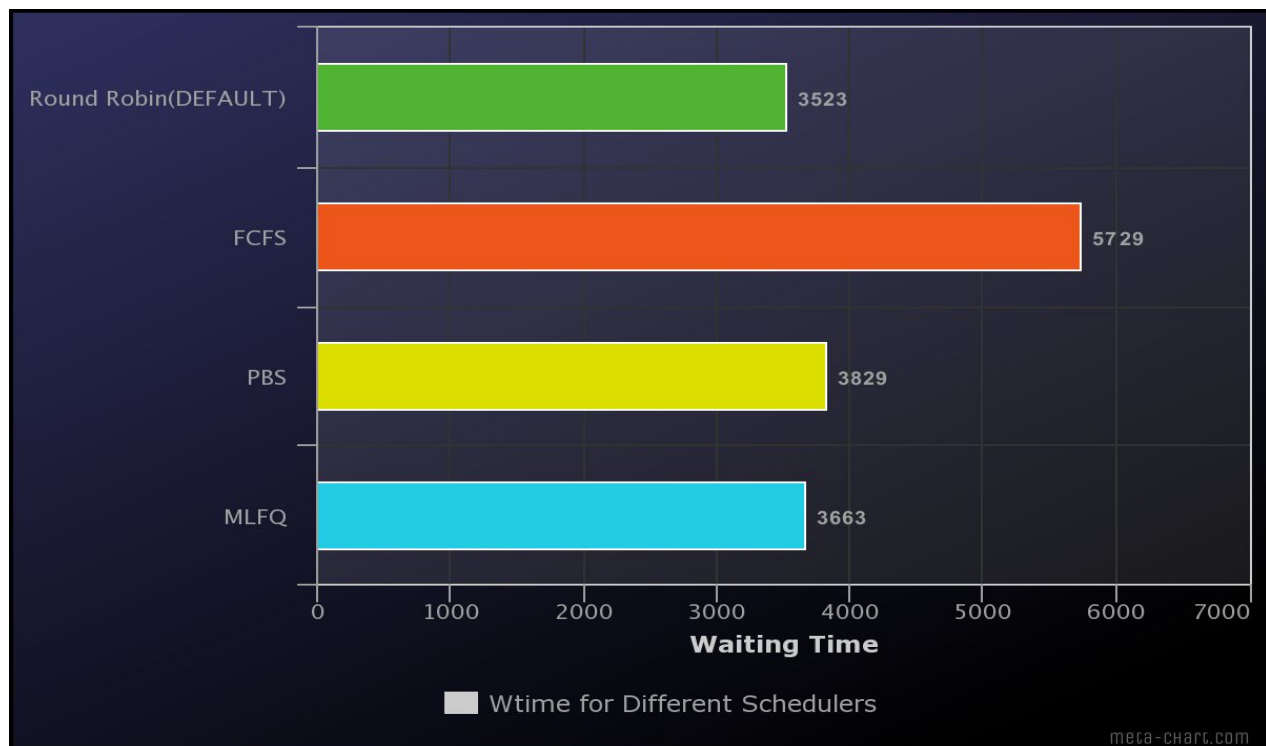


OSNW Assignment 5

Report

Benchmark Results



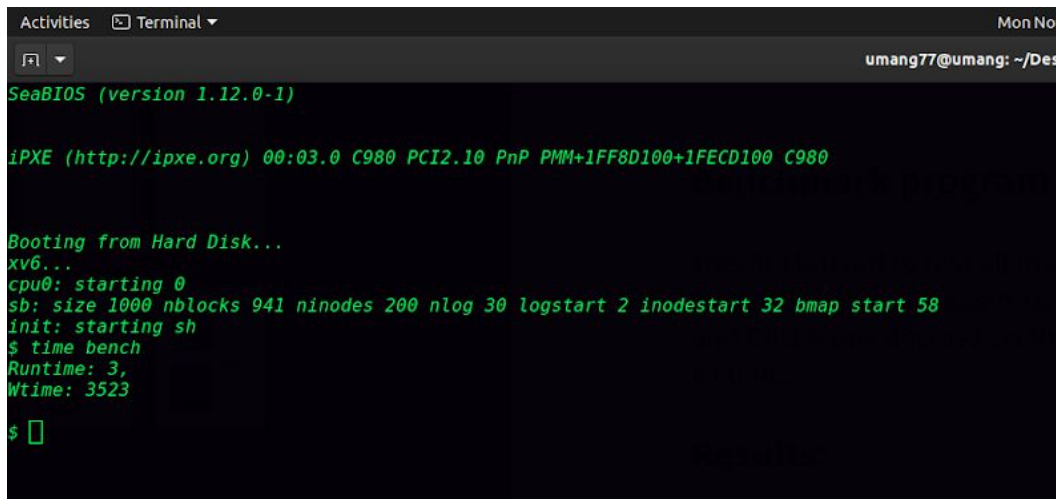
Conclusion

- FCFS has the worst performance of all scheduling policies
 - Round Robin, PBS and MLFQ give somewhat similar result, though on an average, Round Robin works slightly better.
 - Performance: RR > MLFQ > PBS > FCFS.
-

Explanation

1. Round-Robin (DEFAULT)

Round Robin gives the best performance . It is better to adjust the average waiting time desired. All processes had equal priority and so the I/O intensive processes didn't have to wait for CPU intensive processes to finish (Convoy Effect).



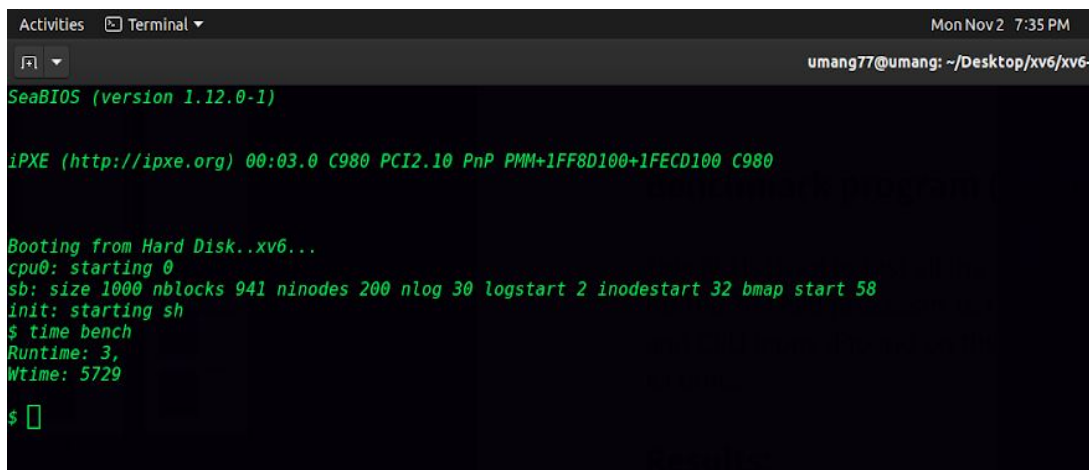
```
Activities Terminal Mon Nov
umang77@umang: ~/Des
SeaBIOS (version 1.12.0-1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8D100+1FECD100 C980

Booting from Hard Disk...
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ time bench
Runtime: 3,
Wtime: 3523
$
```

2. FCFS

FCFS, on an average, is the worst of all scheduling policies and First Come First Serve policy makes long waiting times, even if CPU burst times are short.



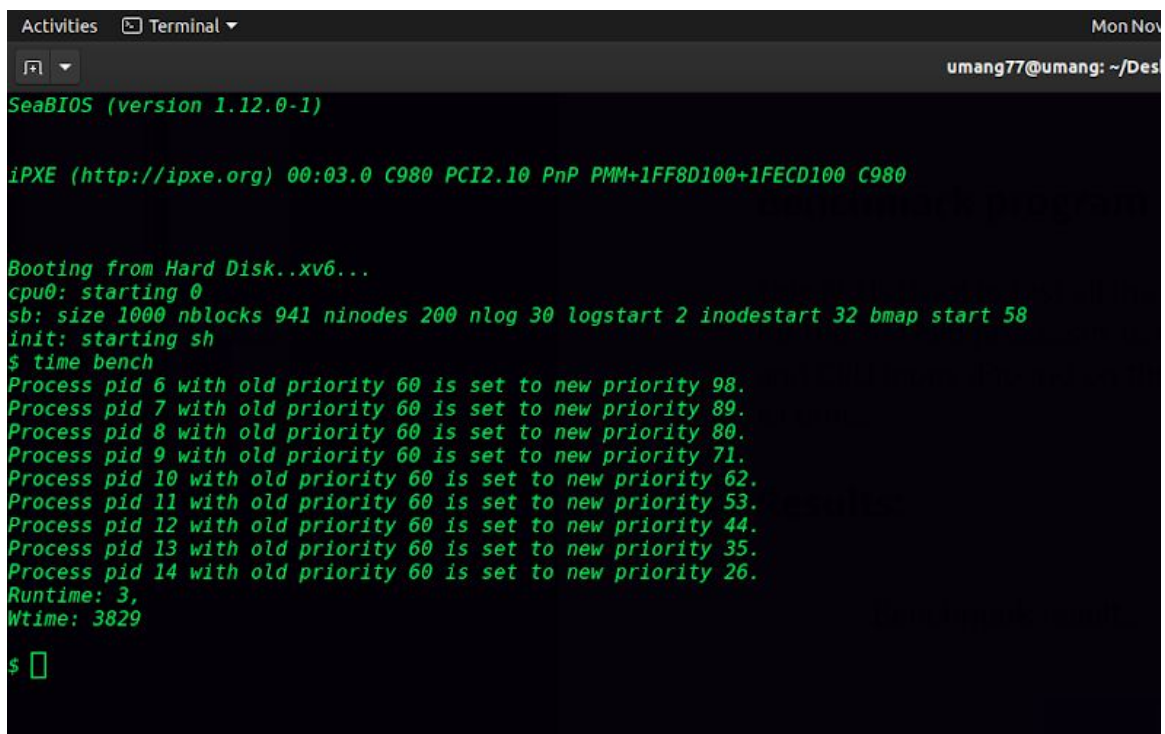
```
Activities Terminal Mon Nov 2 7:35 PM
umang77@umang: ~/Desktop/xv6/xv6
SeaBIOS (version 1.12.0-1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8D100+1FECD100 C980

Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ time bench
Runtime: 3,
Wtime: 5729
$
```

3. PBS

In PBS, I/O intensive processes received lower priority (more Preference) so were executed before CPU intensive processes. It can be that processes may be assigned random priorities and then the output may vary.



```
Activities Terminal Mon Nov
umang77@umang: ~/Des

SeaBIOS (version 1.12.0-1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8D100+1FECDD100 C980

Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ time bench
Process pid 6 with old priority 60 is set to new priority 98.
Process pid 7 with old priority 60 is set to new priority 89.
Process pid 8 with old priority 60 is set to new priority 80.
Process pid 9 with old priority 60 is set to new priority 71.
Process pid 10 with old priority 60 is set to new priority 62.
Process pid 11 with old priority 60 is set to new priority 53.
Process pid 12 with old priority 60 is set to new priority 44.
Process pid 13 with old priority 60 is set to new priority 35.
Process pid 14 with old priority 60 is set to new priority 26.
Runtime: 3,
Wtime: 3829
$
```

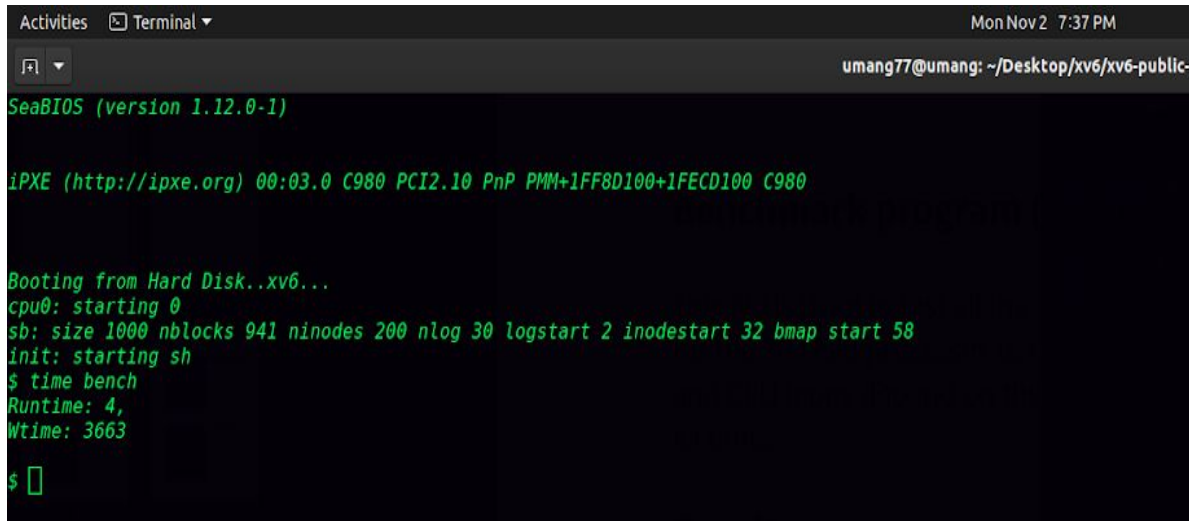
4. MLFQ

In MLFQ, I/O intensive processes were retained in higher priority queues and CPU intensive processes took more ticks so were shifted down to lower priority queues. So, it had better performance than FCFS.

Explanation of point 5 of MLFQ in assignment PDF:

If a process voluntarily relinquishes control of the CPU, it leaves the queuing network, and when the process becomes ready again after the I/O, it is inserted at the tail of the same queue, from which it is relinquished earlier. This can be exploited by a process, as just when the time-slice is about to expire, the process can voluntarily relinquish control of the CPU, and get inserted in the same queue

again. If it ran as normal, then due to time-slice getting expired, it would have been preempted to a lower priority queue. The process, after exploitation, will remain in the higher priority queue, so that it can run again sooner than it should have.



```
Activities Terminal Mon Nov 2 7:37 PM
umang77@umang: ~/Desktop/xv6/xv6-public-
SeaBIOS (version 1.12.0-1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8D100+1FEC100 C980

Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ time bench
Runtime: 4,
Wtime: 3663
$
```

BONUS

Timeline graph for processes running with MLFQ Scheduler.

Y-Axis - Queue Number

X-Axis - time elapsed (in ticks)

