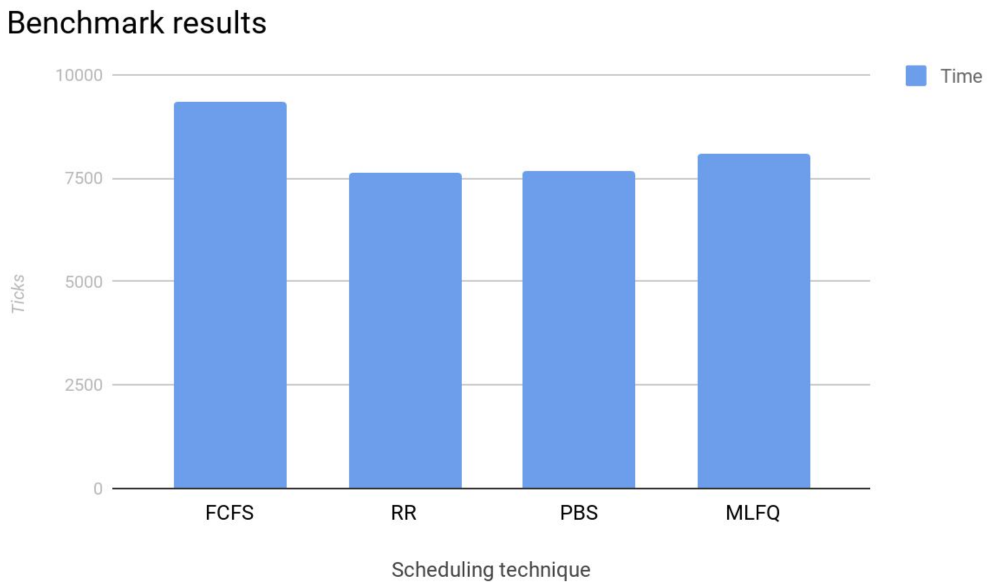
Benchmark program (bench.c):

This file is used to test all the scheduling algorithms. It forks 10 times and the parent waits for the 10 child processes to terminate. The child processes are of different types as the I/O and CPU loops depend on the child number and so there is a good blend of both CPU and IO times.

Results:



Conclusions:

1. FCFS gives the worst performance as it has the highest endtime - creationtime (wtime + rtime).
2. RR, PBS, and MLFQ give somewhat similar results.
3. RR works slightly better than PBS and MLFQ.

   Ranking:

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

Inference:

1. **RR**
   Round Robin performed the **best** as all processes regardless of type (IO or CPU intensive) were given equal priority as they were yielded after every tick. This way the I/O intensive processes didn’t have to wait for CPU intensive processes to finish (Convoy effect).

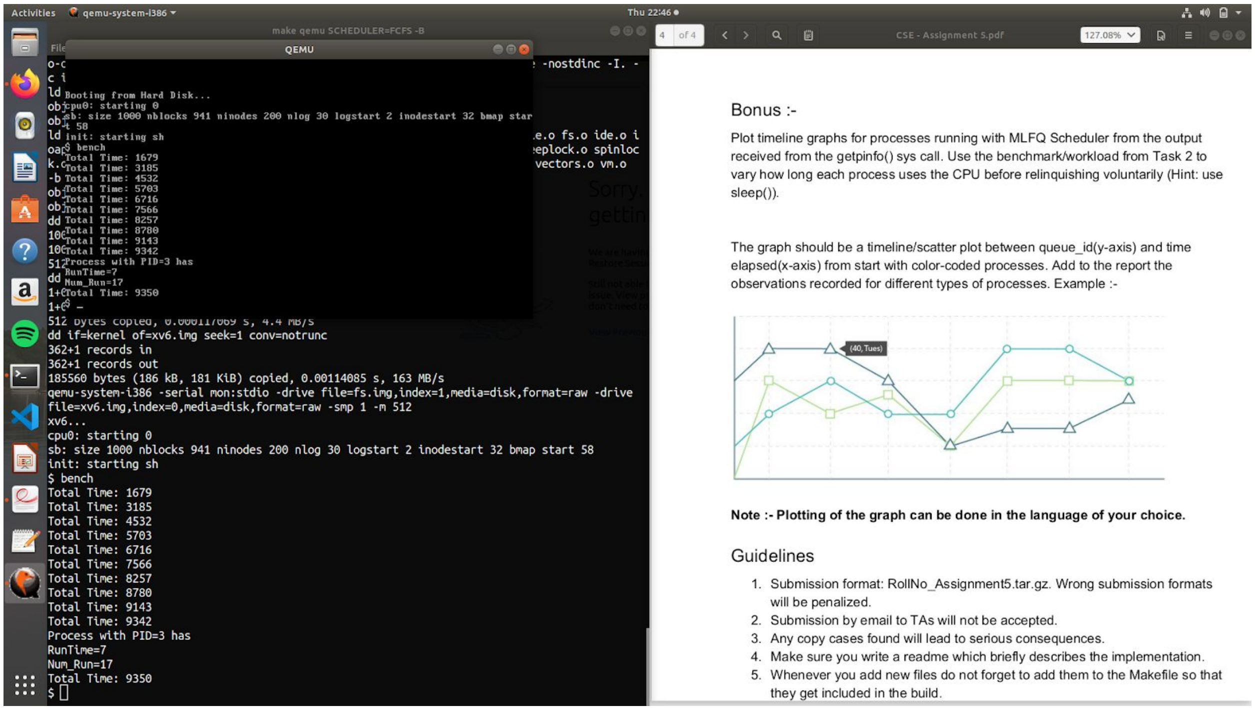
2. **FCFS**
   FCFS performed the **worst** as CPU intensive programs came first and so were served first. The I/O intensive programs had to wait needlessly despite having short CPU burst times.

3. **PBS**
   For Priority based Scheduling the I/O intensive processes received lower priority (more preference), so they were executed before CPU intensive processes when runnable.
   Note: Processes could be assigned priority differently and in that case the outcome would vary.

4. **MLFQ**
   In Multilevel Feedback Queue Scheduling, I/O intensive processes went to sleep (yield) frequently so they were retained in higher priority queues. CPU intensive processes took more ‘ticks’ and hence were constantly shifted down to lower priority queues. Hence, it performed better than FCFS.

Screenshots:

1. FCFS



```
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ bench
Total Time: 1679
Total Time: 3185
Total Time: 4532
Total Time: 5703
Total Time: 6716
Total Time: 7566
Total Time: 8257
Total Time: 8780
Total Time: 9143
Total Time: 9342
Process with PID=3 has
RunTime=7
Num_Run=17
Total Time: 9350
$ _
```

```
512 bytes copied, 0.000117069 s, 4.4 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
362+1 records in
362+1 records out
185560 bytes (186 kB, 181 KiB) copied, 0.00114085 s, 163 MB/s
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive
file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ bench
Total Time: 1679
Total Time: 3185
Total Time: 4532
Total Time: 5703
Total Time: 6716
Total Time: 7566
Total Time: 8257
Total Time: 8780
Total Time: 9143
Total Time: 9342
Process with PID=3 has
RunTime=7
Num_Run=17
Total Time: 9350
$ 
```

Bonus :-

Plot timeline graphs for processes running with MLFQ Scheduler from the output received from the getpinfo() sys call. Use the benchmark/workload from Task 2 to vary how long each process uses the CPU before relinquishing voluntarily (Hint: use sleep()).

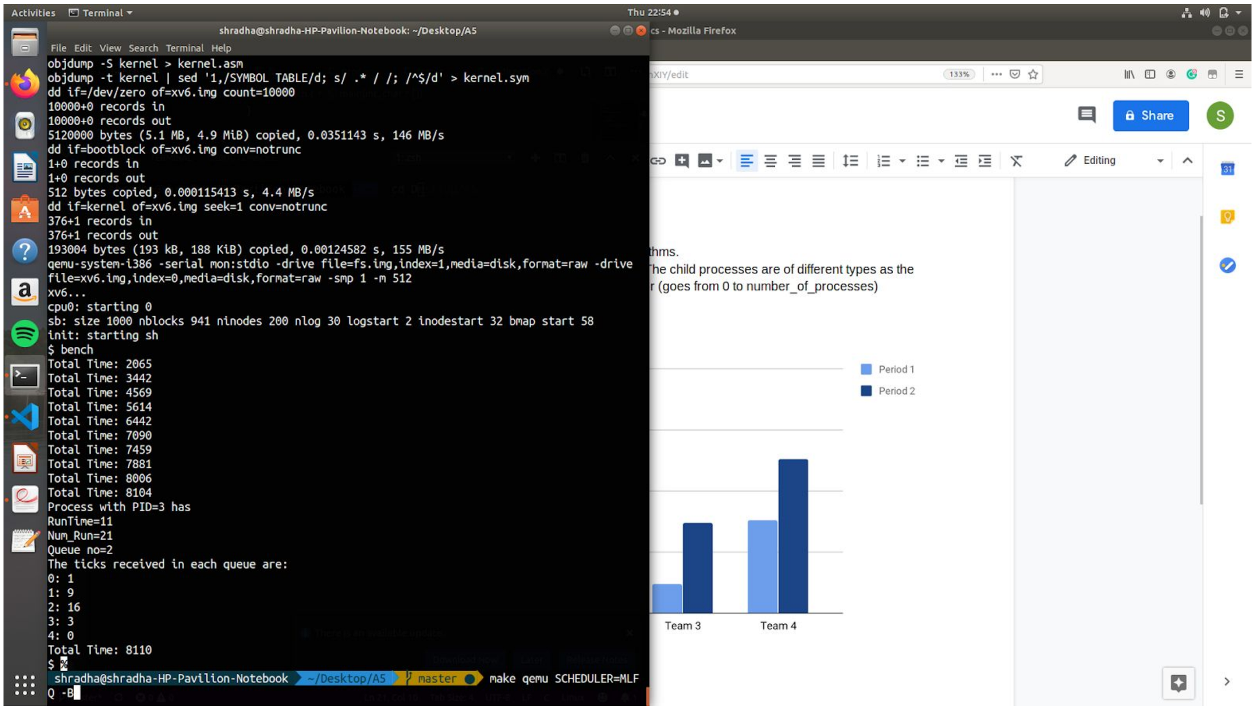
The graph should be a timeline/scatter plot between queue_id(y-axis) and time elapsed(x-axis) from start with color-coded processes. Add to the report the observations recorded for different types of processes. Example :-

Note :- Plotting of the graph can be done in the language of your choice.

Guidelines

1. Submission format: RollNo_Assignment5.tar.gz. Wrong submission formats will be penalized.
2. Submission by email to TAs will not be accepted.
3. Any copy cases found will lead to serious consequences.
4. Make sure you write a readme which briefly describes the implementation.
5. Whenever you add new files do not forget to add them to the Makefile so that they get included in the build.

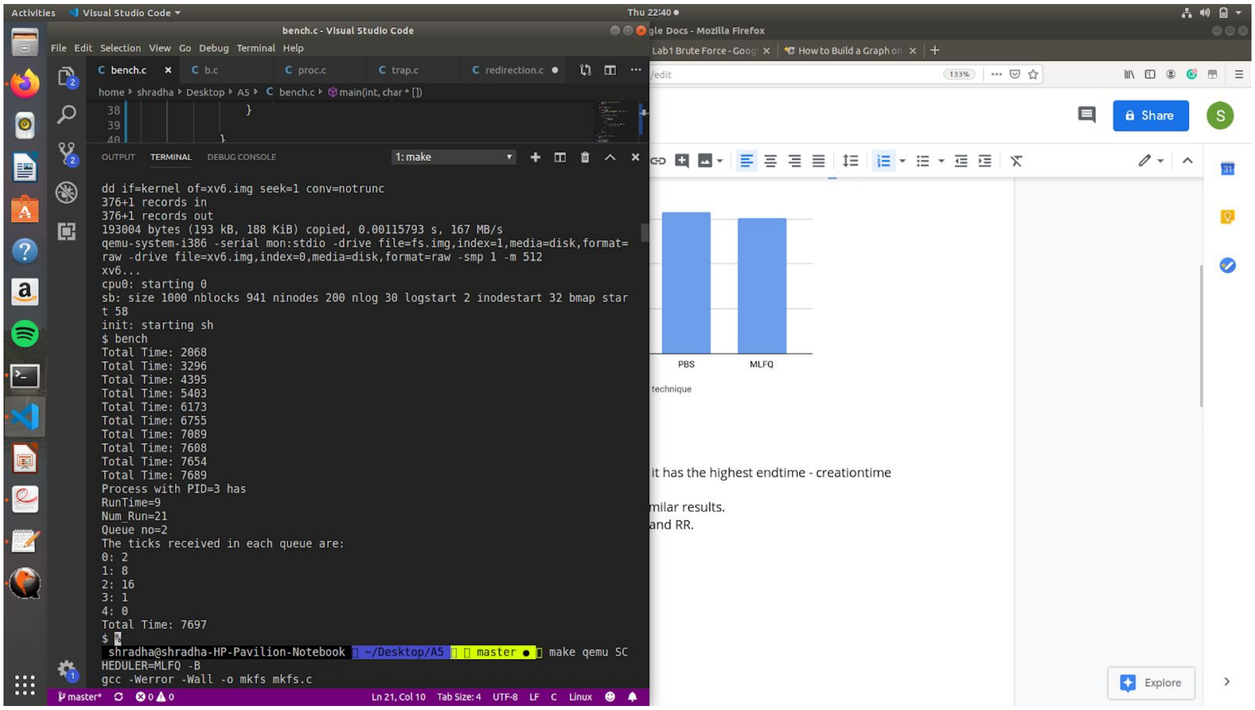
2. MLFQ



```
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0351143 s, 146 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000115413 s, 4.4 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
376+1 records in
376+1 records out
193004 bytes (193 kB, 188 KiB) copied, 0.00124582 s, 155 MB/s
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive
file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ bench
Total Time: 2065
Total Time: 3442
Total Time: 4569
Total Time: 5614
Total Time: 6442
Total Time: 7090
Total Time: 7459
Total Time: 7881
Total Time: 8006
Total Time: 8104
Process with PID=3 has
RunTime=11
Num_Run=21
Queue no=2
The ticks received in each queue are:
0: 1
1: 9
2: 16
3: 3
4: 0
Total Time: 8110
$ %
 shradha@shradha-HP-Pavilion-Notebook  ~/Desktop/A5  master ●  make qemu SCHEDULER=MLF
Q -B
```

thms.
The child processes are of different types as the
r (goes from 0 to number_of_processes)

3. PBS

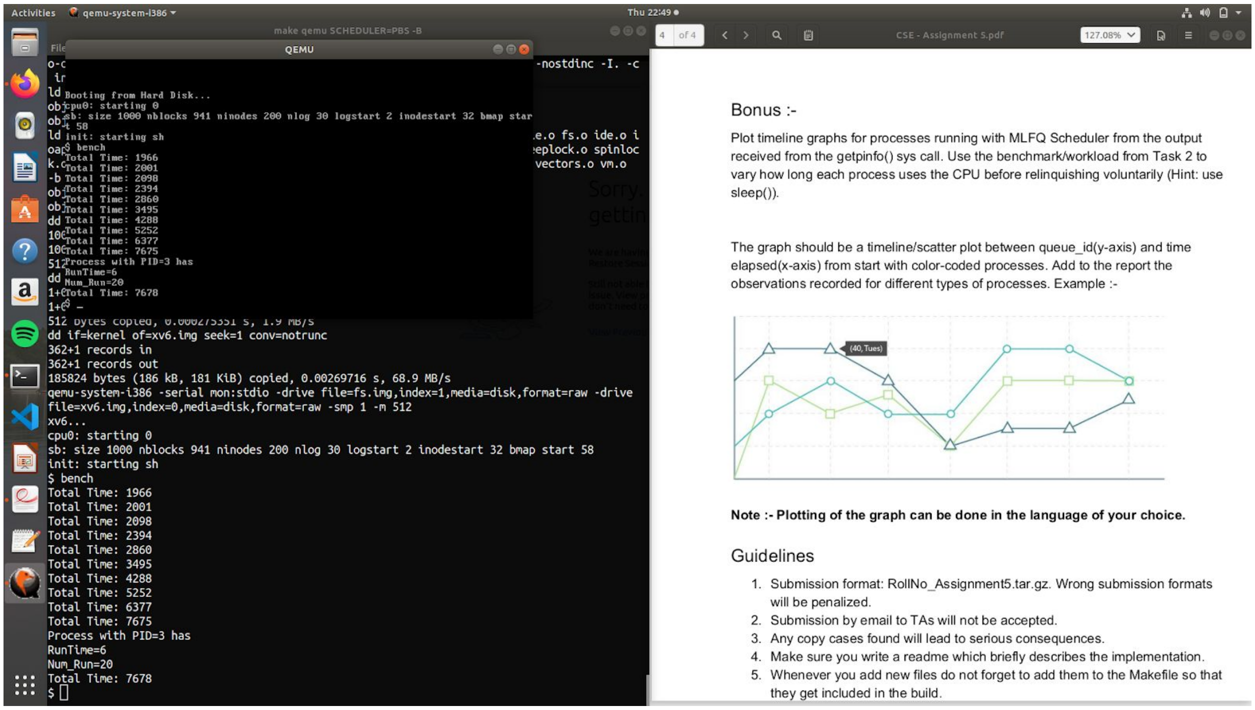


```
dd if=kernel of=xv6.img seek=1 conv=notrunc
376+1 records in
376+1 records out
193004 bytes (193 kB, 188 KiB) copied, 0.00115793 s, 167 MB/s
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=
raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ bench
Total Time: 2068
Total Time: 3296
Total Time: 4395
Total Time: 5403
Total Time: 6173
Total Time: 6755
Total Time: 7089
Total Time: 7608
Total Time: 7654
Total Time: 7689
Process with PID=3 has
RunTime=9
Num_Run=21
Queue no=2
The ticks received in each queue are:
0: 2
1: 8
2: 16
3: 1
4: 0
Total Time: 7697
$ %
 shradha@shradha-HP-Pavilion-Notebook  ~/Desktop/A5   master ●  make qemu SC
HEDULER=MLFQ -B
gcc -Werror -Wall -o mkfs mkfs.c
```

it has the highest endtime - creationtime

milar results.
and RR.

4. RR



```
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ bench
Total Time: 1966
Total Time: 2001
Total Time: 2098
Total Time: 2394
Total Time: 2860
Total Time: 3495
Total Time: 4288
Total Time: 5252
Total Time: 6377
Total Time: 7675
Process with PID=3 has
RunTime=6
Num_Run=20
Total Time: 7678
$ _
```

```
512 bytes copied, 0.000275351 s, 1.9 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
362+1 records in
362+1 records out
185824 bytes (186 kB, 181 KiB) copied, 0.00269716 s, 68.9 MB/s
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive
file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ bench
Total Time: 1966
Total Time: 2001
Total Time: 2098
Total Time: 2394
Total Time: 2860
Total Time: 3495
Total Time: 4288
Total Time: 5252
Total Time: 6377
Total Time: 7675
Process with PID=3 has
RunTime=6
Num_Run=20
Total Time: 7678
$ 
```

Bonus :-

Plot timeline graphs for processes running with MLFQ Scheduler from the output received from the getpinfo() sys call. Use the benchmark/workload from Task 2 to vary how long each process uses the CPU before relinquishing voluntarily (Hint: use sleep()).

The graph should be a timeline/scatter plot between queue_id(y-axis) and time elapsed(x-axis) from start with color-coded processes. Add to the report the observations recorded for different types of processes. Example :-

Note :- Plotting of the graph can be done in the language of your choice.

Guidelines

1. Submission format: RollNo_Assignment5.tar.gz. Wrong submission formats will be penalized.
2. Submission by email to TAs will not be accepted.
3. Any copy cases found will lead to serious consequences.
4. Make sure you write a readme which briefly describes the implementation.
5. Whenever you add new files do not forget to add them to the Makefile so that they get included in the build.