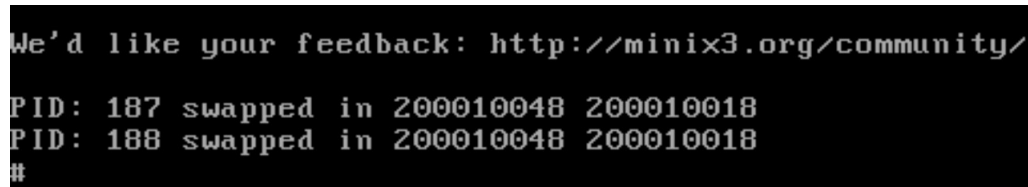


Part 1

In order to perform the given task, the file **schedule.c** was changed at **minix/servers/sched/** location. The following statement was added in schedule.c file at **line number 325** when a process is swapped in:

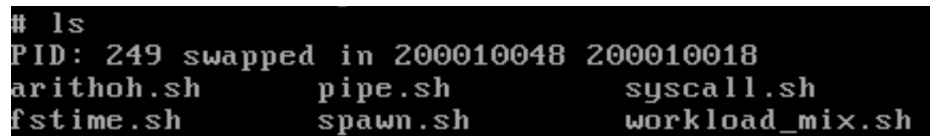
```
1 if(rmp->priority >= USER_Q)
2 {
3     printf("PID: %d swapped in 200010048 200010018\n", _ENDPOINT_P(rmp->endpoint));
4 }
5 return OK;
```

These changes were made in the host machine (Windows). It was transferred to Minix3 Virtual Machine through github. Then, make build MKUPDATE=yes was run and the Minix3 Virtual Machine was rebooted. Upon running the **ls** command, the following output was obtained:



```
We'd like your feedback: http://minix3.org/community/
PID: 187 swapped in 200010048 200010018
PID: 188 swapped in 200010048 200010018
#
```

Figure 1: After Reboot



```
# ls
PID: 249 swapped in 200010048 200010018
arithoh.sh      pipe.sh         syscall.sh
fstime.sh       spawn.sh       workload_mix.sh
```

Figure 2: Sample Command

Part 2

Individual Benchmarks

- We got the source code from the link mentioned in the assignment and copied it to the home folder in the Minix3 VM.
- `gmake` command was run to build the benchmarks.

```
# pwd
/home/OS-Lab-23/Lab_3/byte-unixbench-mod/UnixBench/workload_mix
# ls
PID: 43 swapped in 200010048 200010018
arithoh.sh      pipe.sh        syscall.sh
fstime.sh       spawn.sh       workload_mix.sh
```

Figure 3: Benchmarks

- `arithoh.sh`

```
PID: 73 swapped in 200010048 200010018
PID: 73 swapped in 200010048 200010018
PID: 73 swapped in 200010048 200010018
PID: 73 swapped in 200010048 200010018
PID: 73 swapped in 200010048 200010018
PID: 73 swapped in 200010048 200010018
      14.56 real      14.56 user      0.00 sys
arithoh completed
---
```

Figure 4: `arithoh.sh`

- `fstime.sh`

```
# ./fstime.sh
PID: 74 swapped in 200010048 200010018
PID: 75 swapped in 200010048 200010018
PID: 76 swapped in 200010048 200010018
Write done: 1008000 in 0.8333, score 302400
COUNT:302400:0:KBps
TIME:0.8
Read done: 1000004 in 0.7333, score 340910
COUNT:340910:0:KBps
TIME:0.7
PID: 76 swapped in 200010048 200010018
Copy done: 1000004 in 1.6667, score 150000
COUNT:150000:0:KBps
TIME:1.7
      14.25 real      0.20 user      3.05 sys
fstime completed
---
```

Figure 5: `fstime.sh`

- syscall.sh

```
# ./syscall.sh
PID: 77 swapped in 200010048 200010018
PID: 78 swapped in 200010048 200010018
PID: 79 swapped in 200010048 200010018
PID: 79 swapped in 200010048 200010018
PID: 79 swapped in 200010048 200010018
PID: 79 swapped in 200010048 200010018
PID: 79 swapped in 200010048 200010018
      4.45 real      1.53 user      2.91 sys
syscall completed
---
```

Figure 6: syscall.sh

- pipe.sh

```
# ./pipe.sh
PID: 80 swapped in 200010048 200010018
PID: 81 swapped in 200010048 200010018
PID: 82 swapped in 200010048 200010018
PID: 82 swapped in 200010048 200010018
PID: 82 swapped in 200010048 200010018
      6.40 real      0.66 user      5.73 sys
pipe completed
---
```

Figure 7: pipe.sh

- spawn.sh

```
PID: 238 swapped in 200010048 200010018
PID: 239 swapped in 200010048 200010018
PID: 240 swapped in 200010048 200010018
PID: 241 swapped in 200010048 200010018
PID: 242 swapped in 200010048 200010018
PID: 243 swapped in 200010048 200010018
PID: 244 swapped in 200010048 200010018
PID: 245 swapped in 200010048 200010018
      5.46 real      0.21 user      4.96 sys
spawn completed
```

Figure 8: spawn.sh

Plots

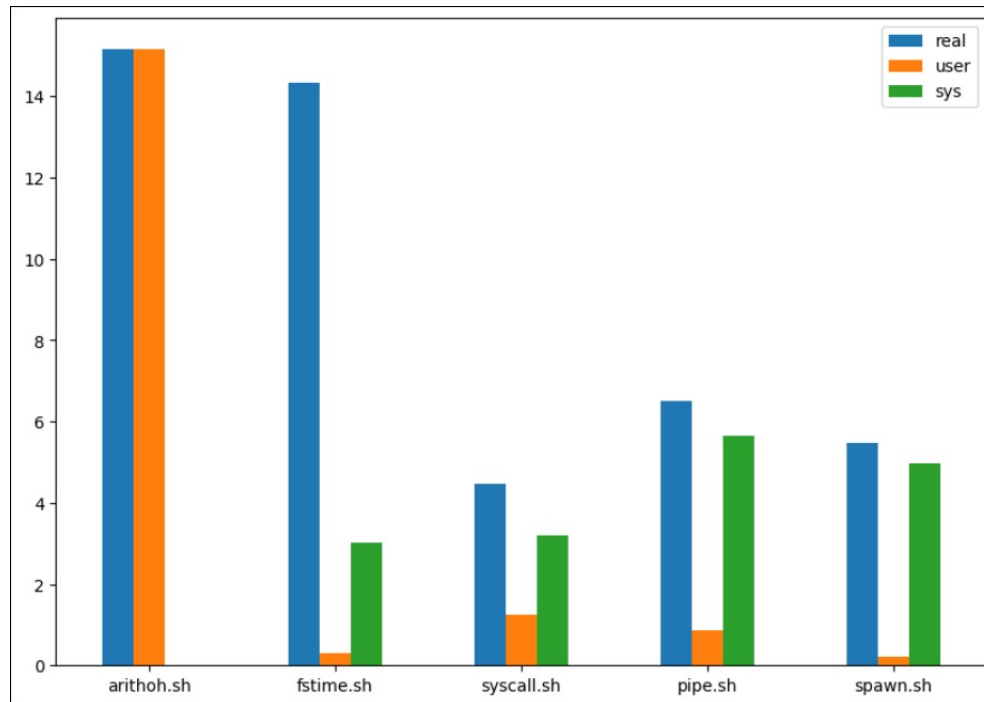


Figure 9: Plots comparing the real, user and system times for above processes

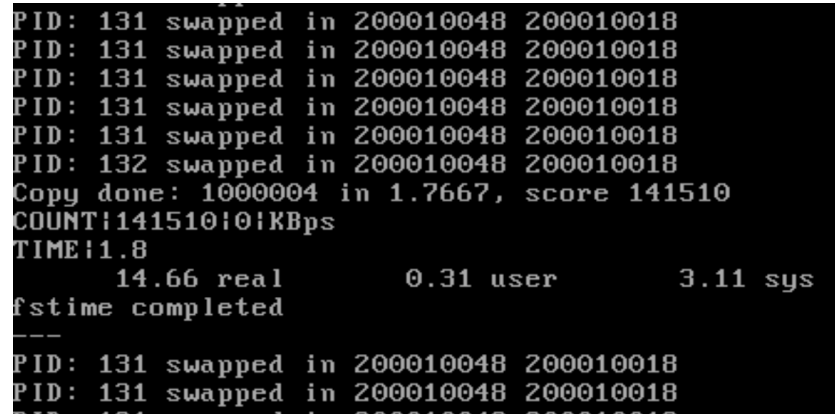
Inferences

- **arithoh.sh:** It is observed that there is no system time used. The entire process is run in user mode as they are cpu intensive processes. Since it is a CPU bound process it has a lower priority hence it's preemption frequency is high.
- **ftime.sh:** This is an I/O intensive process. Here, the real time is high as compared to user and sys time because the process consists of several calls to time, date, sleep etc. (real time processes whose deadlines have to be met).
- **syscall.sh, pipe.sh, spawn.sh:** These benchmarks mostly consist of system calls. Hence, we can see that most of the process is run in kernel mode rather than user mode. Their preemption frequency is lower as switching between user and kernel mode has an overhead.

Workload Mix Benchmarks

- arithoh.sh & fstime.sh

```
1 #!/bin/sh
2 ./arithoh.sh &
3 ./fstime.sh &
4 wait
```



```
PID: 131 swapped in 200010048 200010018
PID: 131 swapped in 200010048 200010018
PID: 131 swapped in 200010048 200010018
PID: 131 swapped in 200010048 200010018
PID: 131 swapped in 200010048 200010018
PID: 132 swapped in 200010048 200010018
Copy done: 1000004 in 1.7667, score 141510
COUNT:141510:0:KBps
TIME:1.8
      14.66 real          0.31 user          3.11 sys
fstime completed
---
PID: 131 swapped in 200010048 200010018
PID: 131 swapped in 200010048 200010018
```

Figure 10: ari_fs.sh

Inferences : It is observed that, fstime completes its execution before arithoh as fstime mostly consists of **real time processes** which have the **highest priority** when compared to arith operations. Also among the two processes, **arithoh.sh** (PID: 131) has a higher pre-emption frequency as it is a lower priority process. The **fstime.sh** process is switched back to user mode only when all the I/O routines are completed (PID: 132). *This is visible in the figure.*

- Our Benchmarks : loop.c

```
1 int main(argc, argv)
2 int argc;
3 char *argv[];
4 {
5     while(1)
6     {
7     }
8
9 return(0);
10 }
```

```
1 #!/bin/sh
2 ./loop.sh &
3 ./arithoh.sh &
4 wait
```

```
PID: 196 swapped in 200010048 200010018
PID: 196 swapped in 200010048 200010018
PID: 195 swapped in 200010048 200010018
PID: 195 swapped in 200010048 200010018
PID: 196 swapped in 200010048 200010018
PID: 196 swapped in 200010048 200010018
PID: 195 swapped in 200010048 200010018
PID: 196 swapped in 200010048 200010018
PID: 196 swapped in 200010048 200010018
```

Figure 11: loop_arith.sh

Inferences : We observe that even though **loop.c** consists of an infinite loop and will never complete execution, the scheduler gives CPU time to the arithmetic process in order to complete its execution. The scheduler alternates between the two processes. Hence, the scheduler does not wait for a process to be completed in order to run the next one. This is visible in the figure.