Assignment 3

Operating Systems Lab

Mohammad Sameer | 190010024 | 19010024@iitdh.ac.in

Part 1

- In part 1 I have modified the Minix 3 source code such that the string `PID <pid> swapped in` is printed, whenever a user-level process is brought in by the scheduler.
- To do this, I have added the following lines of C code in to the `schedule.c` file, located at `/minix/minix/servers/sched` directory.

```
if(rmp->priority >= USER_Q){
    printf("Minix: PID %d swapped in\n", _ENDPOINT_P(rmp->endpoint));
}
```

- These lines of code were added to `schedule_process` function present in the said file.
- To make appropriate changes I have written a `run.sh` bash file.
- Which when run, copies the modified `schedule.c` file into appropriate location and rebuilds the minix OS.
- Upon successful build and reboot the changes are reflected.

```
Minix: PID 367 created
Minix: PID 367 exited
Minix: PID 366 exited
Minix: PID 365 exited
Minix: PID 368 created
Minix: PID 369 created
Minix: PID 370 created
Minix: PID 370 exited
Minix: PID 369 exited
Minix: PID 368 exited
ls
Minix: PID 371 created
Makefile   bin        lib        man        run        share      tmp
adm        etc        libdata    mdec       run.sh     spool      var
ast        games      libexec    pkg        sbin       src
benchmarks include    log        preserve   schedule.c tests
Minix: PID 371 exited
# ls
Minix: PID 372 created
Makefile   bin        lib        man        run        share      tmp
adm        etc        libdata    mdec       run.sh     spool      var
ast        games      libexec    pkg        sbin       src
benchmarks include    log        preserve   schedule.c tests
Minix: PID 372 exited
# bash run.sh
```

Figure 1: Copied files and command to run `run.sh`

```
    install  //usr/lib/fonts
do-hdboot ===> releasetools
install -N /usr/src/etc -c -p -r ../minix/servers/ds/ds /boot/minix/.temp/mod01_
ds
install -N /usr/src/etc -c -p -r ../minix/servers/rs/rs /boot/minix/.temp/mod02_
rs
install -N /usr/src/etc -c -p -r ../minix/servers/pm/pm /boot/minix/.temp/mod03_
pm
install -N /usr/src/etc -c -p -r ../minix/servers/sched/sched /boot/minix/.temp/
mod04_sched
install -N /usr/src/etc -c -p -r ../minix/servers/vfs/vfs /boot/minix/.temp/mod0
5_vfs
install -N /usr/src/etc -c -p -r ../minix/drivers/storage/memory/memory /boot/mi
nix/.temp/mod06_memory
install -N /usr/src/etc -c -p -r ../minix/drivers/tty/tty/tty /boot/minix/.temp/
mod07_tty
install -N /usr/src/etc -c -p -r ../minix/fs/mfs/mfs /boot/minix/.temp/mod08_mfs
install -N /usr/src/etc -c -p -r ../minix/servers/vm/vm /boot/minix/.temp/mod09_
vm
install -N /usr/src/etc -c -p -r ../minix/fs/pfs/pfs /boot/minix/.temp/mod10_pfs
install -N /usr/src/etc -c -p -r ../sbin/init/init /boot/minix/.temp/mod11_init
Done.
Build started at:  Sun Jan 30 03:43:30 IST 2022
Build finished at: Sun Jan 30 03:45:52 IST 2022
```

Figure 2: Successful Build

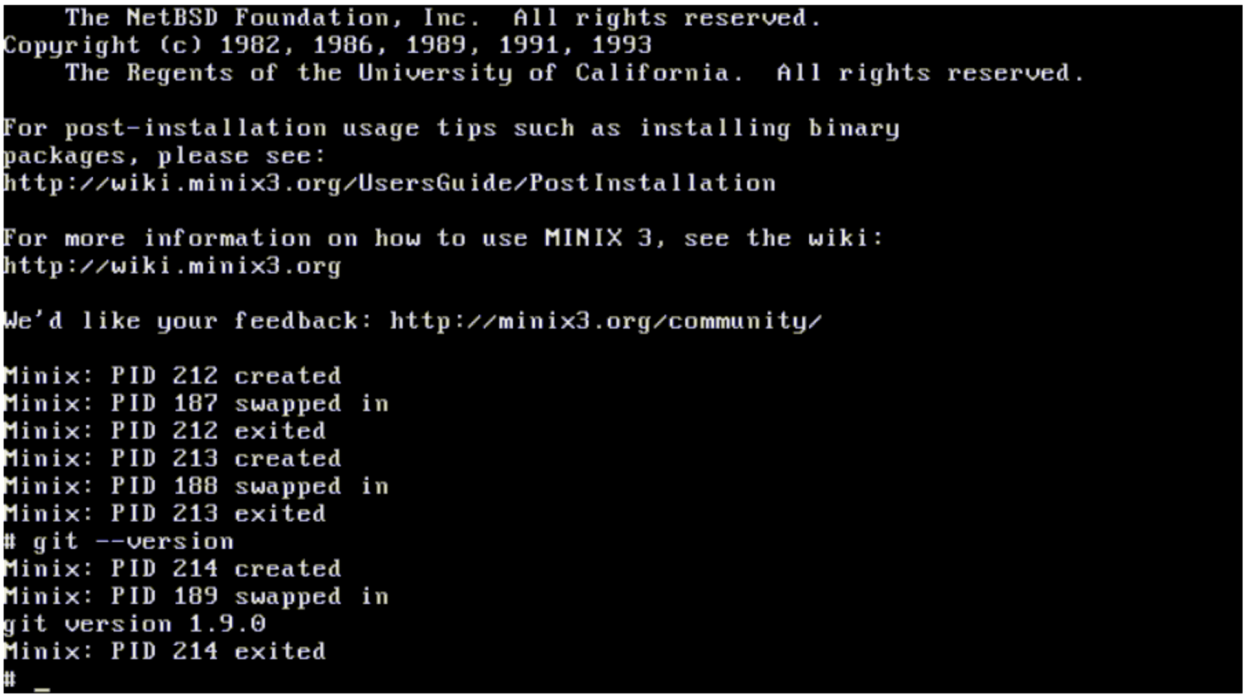

Figure 3: Changes after reboot

Part 2

In this part using UnixBench benchmark suite, I have run various combination of workloads to understand and analyze the minix scheduler and schedule orders. Below section explain the various workloads used.

2.1 `workload_mix1.sh`

The below is the content of `workload_mix1.sh` file,

```
#!/bin/sh
./arithoh.sh &
./fstime.sh &
wait
```

- In this workload I have used two workloads, namely `arithoh.sh` and `fstime.sh`.
- After looking at their source and the order of scheduling, I came to know that `arithoh.sh` is CPU bound, it does some CPU bound arithmetic calculations where as `fstime.sh` is I/O bound, it reads and writes some buffer data into the memory.

```
fstime.sh           syscall.sh          workload_mix2.sh    workload_mix5.sh
pipe.sh             workload_mix.sh     workload_mix3.sh
Minix: PID 218 exited
# ./workload_mix1.sh
Minix: PID 219 created
Minix: PID 194 swapped in
Minix: PID 220 created
Minix: PID 195 swapped in
Minix: PID 221 created
Minix: PID 196 swapped in
Minix: PID 222 created
Minix: PID 197 swapped in
Minix: PID 223 created
Minix: PID 198 swapped in
Minix: PID 224 created
Minix: PID 199 swapped in
Minix: PID 225 created
Minix: PID 200 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
```

Figure 4: Started to run `workload_mix1.sh`

```
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 200 swapped in
Copy done: 1000004 in 2.0667, score 120968
COUNT|120968|0|KBps
TIME|2.1
Minix: PID 225 exited
      15.33 real       0.33 user       3.76 sys
Minix: PID 223 exited
fstime completed
---
Minix: PID 221 exited
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
Minix: PID 199 swapped in
```

Figure 5: `fstime.sh` completed

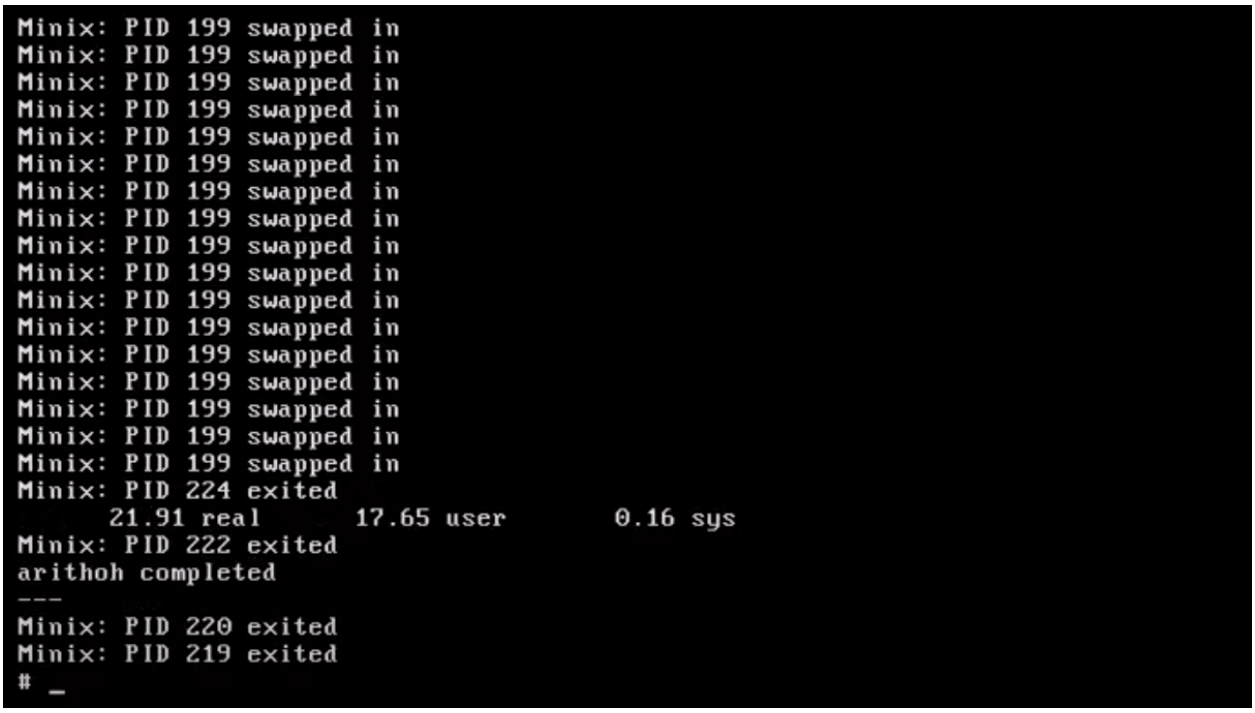

Figure 6: `arithoh.sh` completed

- From the figure 4, we can see that `arithoh.sh`, with `PID 199`, being swapped in while `fstime.sh`, with `PID 200` is waiting for I/O response.
- When `fstime.sh` becomes ready after I/O response, it is scheduled since it has relatively smaller burst time and it is completed as seen in figure 5.
- After this the `arithoh.sh` is scheduled until completion as seen from figure 6.

2.2 `workload_mix2.sh`

The below is the content of `workload_mix2.sh` file,

```
#!/bin/sh
./arithoh.sh &
./syscall.sh &
wait
```

- In this workload I have used two workloads, namely `arithoh.sh` and `syscall.sh`.
- After looking at their source and the order of scheduling, I came to know that both `arithoh.sh` and `syscall.sh` are CPU bound.
- The `arithoh.sh` does some CPU bound arithmetic calculations where as `syscall.sh` sits in a loop calling the system.

```
pipe.sh             workload_mix.sh     workload_mix3.sh
Minix: PID 226 exited
# ./workload_mix2.sh
Minix: PID 227 created
Minix: PID 202 swapped in
Minix: PID 228 created
Minix: PID 203 swapped in
Minix: PID 229 created
Minix: PID 204 swapped in
Minix: PID 230 created
Minix: PID 205 swapped in
Minix: PID 231 created
Minix: PID 206 swapped in
Minix: PID 232 created
Minix: PID 207 swapped in
Minix: PID 233 created
Minix: PID 208 swapped in
Minix: PID 207 swapped in
Minix: PID 208 swapped in
Minix: PID 207 swapped in
Minix: PID 208 swapped in
Minix: PID 207 swapped in
Minix: PID 208 swapped in
Minix: PID 207 swapped in
```

Figure 7: Started to run `workload_mix2.sh`

```
Minix: PID 207 swapped in
Minix: PID 208 swapped in
Minix: PID 208 swapped in
Minix: PID 207 swapped in
Minix: PID 208 swapped in
Minix: PID 207 swapped in
Minix: PID 233 exited
       6.98 real       1.61 user       4.00 sys
Minix: PID 231 exited
syscall completed
---
Minix: PID 229 exited
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
Minix: PID 207 swapped in
```

Figure 8: `syscall.sh` completed

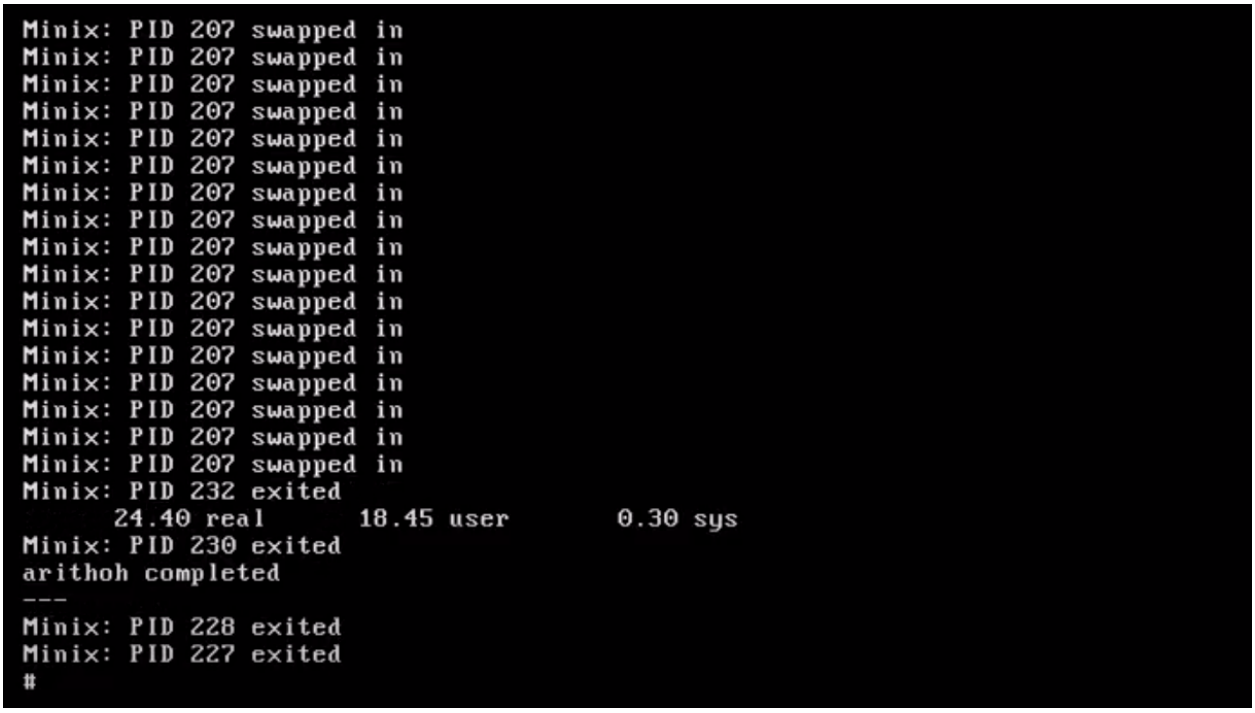

Figure 9: `arithoh.sh` completed

- From figure 7, we can see that `arithoh.sh`, with `PID 207` and `syscall.sh`, with `PID 208` being swapped one after the other depending upon their priority.
- Then, `syscall.sh` completes first as it seems to less CPU intensive than `arithoh.sh` as seen from figure 8.
- After this, the `arithoh.sh` is scheduled until completion as seen from figure 9.

2.3 `workload_mix3.sh`

The below is the content of `workload_mix3.sh` file,

```
#!/bin/sh
./arithoh.sh &
./spawn.sh &
wait
```

- In this workload I have used two workloads, namely `arithoh.sh` and `spawn.sh`.
- After looking at their source and the order of scheduling, I came to know that both `arithoh.sh` and `spawn.sh` are CPU bound.
- The `arithoh.sh` does some CPU bound arithmetic calculations where as `spawn.sh` keeps creating (forks) new process which are exited immediately.

```
Minix: PID 936 exited
Minix: PID 937 created
Minix: PID 62 swapped in
Minix: PID 937 exited
Minix: PID 938 created
Minix: PID 63 swapped in
Minix: PID 938 exited
Minix: PID 939 created
Minix: PID 64 swapped in
Minix: PID 939 exited
Minix: PID 940 created
Minix: PID 65 swapped in
Minix: PID 940 exited
Minix: PID 941 created
Minix: PID 66 swapped in
Minix: PID 941 exited
Minix: PID 942 created
Minix: PID 67 swapped in
Minix: PID 942 exited
Minix: PID 943 created
Minix: PID 68 swapped in
Minix: PID 943 exited
Minix: PID 944 created
Minix: PID 69 swapped in
Minix: PID 944 exited
```

Figure 10: Running `workload_mix3.sh`

```
Minix: PID 230 swapped in
Minix: PID 10239 exited
Minix: PID 10240 created
Minix: PID 231 swapped in
Minix: PID 10240 exited
Minix: PID 10241 created
Minix: PID 232 swapped in
Minix: PID 10241 exited
Minix: PID 10242 created
Minix: PID 233 swapped in
Minix: PID 10242 exited
Minix: PID 241 exited
      18.30 real       0.26 user      10.20 sys
Minix: PID 239 exited
spawn completed
---
Minix: PID 237 exited
Minix: PID 215 swapped in
Minix: PID 215 swapped in
Minix: PID 215 swapped in
Minix: PID 215 swapped in
Minix: PID 215 swapped in
Minix: PID 215 swapped in
Minix: PID 215 swapped in
```

Figure 11: `spawn.sh` completed

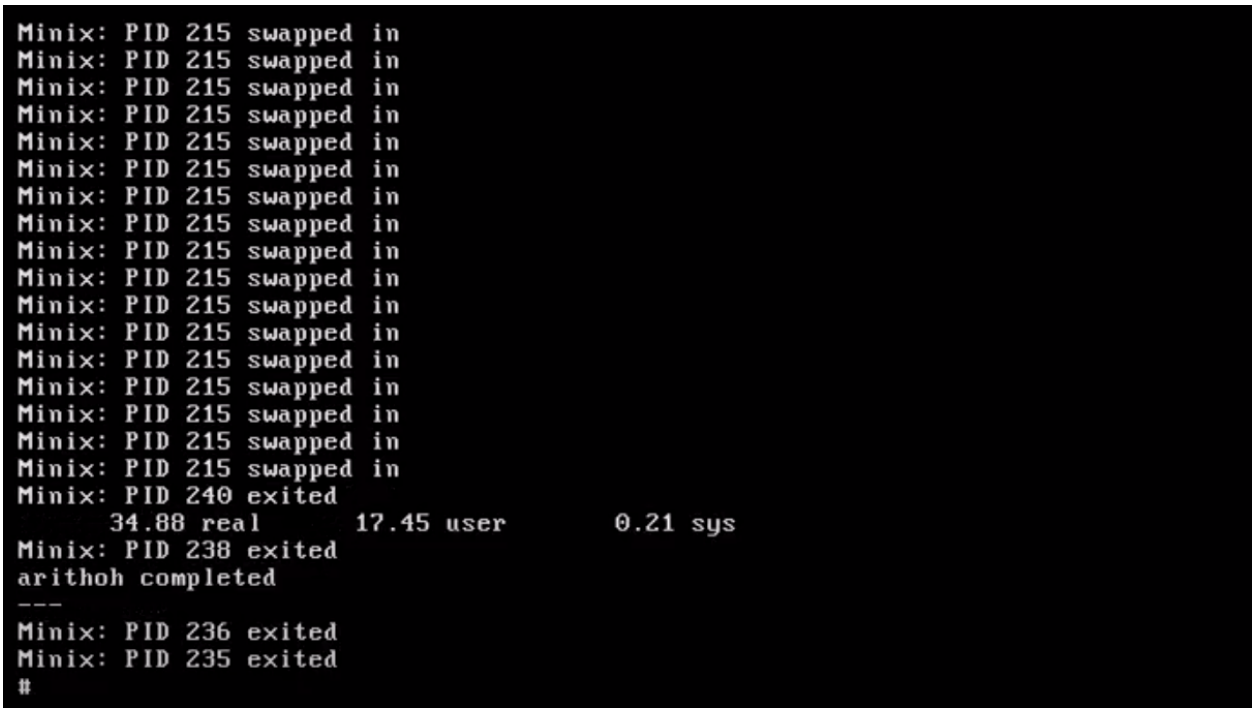

Figure 12: `arithoh.sh` completed

- From the figure 10, we can see that new process are continuously created by `spawn.sh` and are being scheduled as well to minimize the response time. These newly created process exit immediately, which is observed from the source code.
- After some time, the `spawn.sh` along with the newly created process terminate as seen from figure 11.
- Then, the `arithoh.sh`, with `PID 215` is scheduled until completion as seen from figure 12.

2.4 `workload_mix4.sh`

The below is the content of `workload_mix4.sh` file,

```
#!/bin/sh
./fstime.sh &
./pipe.sh &
wait
```

- In this workload I have used two workloads, namely `fstime.sh` and `pipe.sh`.
- After looking at their source and the order of scheduling, I came to know that both `fstime.sh` and `pipe.sh` are I/O bound.

- The `fstime.sh` reads and writes some buffer data into the memory. where as `pipe.sh` test single process pipe throughput (no context switching).

```
ls
Minix: PID 10243 created
Minix: PID 234 swapped in
arithoh.sh          spawn.sh            workload_mix1.sh    workload_mix4.sh
fstime.sh           syscall.sh          workload_mix2.sh    workload_mix5.sh
pipe.sh             workload_mix.sh     workload_mix3.sh
Minix: PID 10243 exited
# ./workload_mix4.sh
Minix: PID 10244 created
Minix: PID 235 swapped in
Minix: PID 10245 created
Minix: PID 236 swapped in
Minix: PID 10246 created
Minix: PID 237 swapped in
Minix: PID 10247 created
Minix: PID 238 swapped in
Minix: PID 10248 created
Minix: PID 239 swapped in
Minix: PID 10249 created
Minix: PID 240 swapped in
Minix: PID 10250 created
Minix: PID 241 swapped in
Minix: PID 241 swapped in
Minix: PID 241 swapped in
```

Figure 13: Running `workload_mix4.sh`

```
Minix: PID 10245 created
Minix: PID 236 swapped in
Minix: PID 10246 created
Minix: PID 237 swapped in
Minix: PID 10247 created
Minix: PID 238 swapped in
Minix: PID 10248 created
Minix: PID 239 swapped in
Minix: PID 10249 created
Minix: PID 240 swapped in
Minix: PID 10250 created
Minix: PID 241 swapped in
Minix: PID 241 swapped in
Minix: PID 241 swapped in
Write done: 1008000 in 1.6167, score 155876
COUNT¦155876¦0¦KBps
TIME¦1.6
Minix: PID 241 swapped in
Minix: PID 10250 exited
       9.03 real       0.83 user       7.23 sys
Minix: PID 10248 exited
pipe completed
---
Minix: PID 10246 exited
```

Figure 14: `pipe.sh` completed

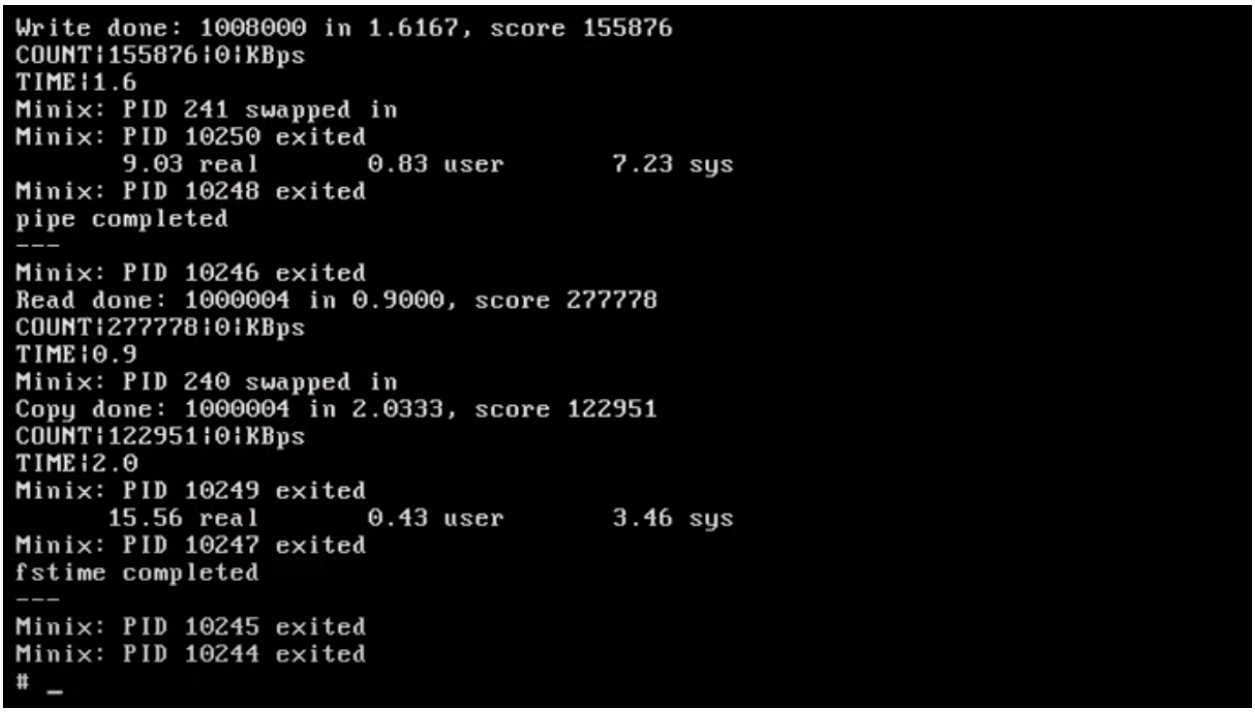

Figure 15: `fstime.sh` completed

- From the figure 13, we can `fstime.sh`, with `PID 240` is waiting for I/O response, while `pipe.sh`, with `PID 241` is swapped in. From the source code, it was observed that `pipe.sh` has no context switch.
- So once `pipe.sh` is swapped in, it runs until completion as seen from figure 14.
- After this, `fstime.sh` is swapped in and runs until completion as seen from figure 15.

2.5 `workload_mix5.sh`

The below is the content of `workload_mix5.sh` file,

```
#!/bin/sh
./arithoh.sh &
./arithoh.sh &
./arithoh.sh &
wait
```

- In this workload I have used a single workload, namely `arithoh.sh` but ran it with 3 different instances.
- After looking at their source and the order of scheduling, I came to know that both `arithoh.sh` is CPU bound.

- The `arithoh.sh` does some CPU bound arithmetic calculations.

```
fstime.sh           syscall.sh          workload_mix2.sh    workload_mix5.sh
pipe.sh             workload_mix.sh     workload_mix3.sh
Minix: PID 10263 exited
# ./workload_mix5.sh
Minix: PID 10264 created
Minix: PID 255 swapped in
Minix: PID 10265 created
Minix: PID 7 swapped in
Minix: PID 10266 created
Minix: PID 11 swapped in
Minix: PID 10267 created
Minix: PID 15 swapped in
Minix: PID 10268 created
Minix: PID 18 swapped in
Minix: PID 10269 created
Minix: PID 33 swapped in
Minix: PID 10270 created
Minix: PID 35 swapped in
Minix: PID 10271 created
Minix: PID 36 swapped in
Minix: PID 10272 created
Minix: PID 37 swapped in
Minix: PID 10273 created
Minix: PID 38 swapped in
```

Figure 16: Starting to run `workload_mix5.sh`

```
Minix: PID 36 swapped in
Minix: PID 36 swapped in
Minix: PID 37 swapped in
Minix: PID 38 swapped in
Minix: PID 37 swapped in
Minix: PID 38 swapped in
Minix: PID 36 swapped in
Minix: PID 37 swapped in
Minix: PID 38 swapped in
Minix: PID 38 swapped in
Minix: PID 36 swapped in
Minix: PID 37 swapped in
Minix: PID 38 swapped in
Minix: PID 36 swapped in
Minix: PID 37 swapped in
Minix: PID 36 swapped in
Minix: PID 37 swapped in
Minix: PID 38 swapped in
Minix: PID 37 swapped in
Minix: PID 38 swapped in
Minix: PID 36 swapped in
Minix: PID 38 swapped in
Minix: PID 36 swapped in
Minix: PID 37 swapped in
```

Figure 17: During `workload_mix5.sh` is running

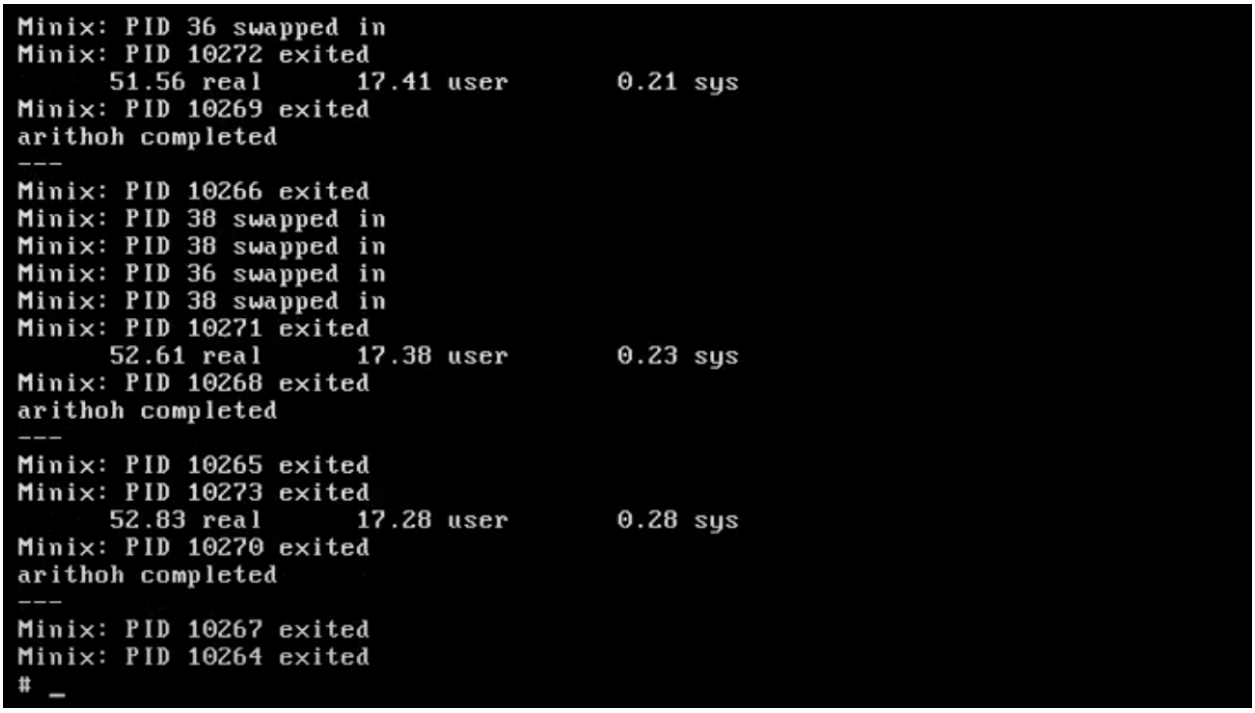

Figure 18: All `arithoh.sh` completed

- From figure 16 and 17, we can see that the different instances of `arithoh.sh` have `PID 36, 37 and 38`.
- From figure 17, we can see that all the CPU bound `arithoh.sh` processes are run in alternating manner depending upon their priority.
- From figure 18, we can observe that all the `arithoh.sh` processes complete one after the other.