

# CS314 : Operating Systems Lab

## Lab 5

Sourabh Bhosale (200010004)

Dibyashu Kashyap (200010013)

February 5, 2023

# 1 Part 1

## Problem

Prepare (at least 4) workload mixes having different characteristics, ranging from all compute-intensive benchmarks to all I/O-intensive benchmarks. Each workload should spawn around 5 processes. You can compose these workloads of benchmarks from the UnixBench suite, or write your own.

## Answer

As the problem required us to encompass the time quanta spent in the CPU by the processes. For that, file `system.c` at location `minix/kernel/` was changed slightly. In the function `sched_proc()`, the quanta allotted and used were extracted as follows:

```
printf("Minix : Allotted Quantum: %d ms\n",  
       Minix : Used Quantum: %d ms\n",  
       p->p_quantum_size_ms, p->p_quantum_size_ms -  
       cpu_time_2_ms(p->p_cpu_time_left));
```

To make appropriate changes we have written a `runme1.sh` file, which when run, copies the modified `system.c` file into appropriate location and rebuilds the minix OS. Upon executing `runme1.sh`, and successful build followed by a reboot, the changes should be reflected. The workloads used by us and our observations and inferences are presented in the sections below.

```
#!/bin/bash  
  
cp system.c /usr/src/minix/kernel/system.c;  
cd ../usr/src;  
make build MKUPDATE=yes > log.txt;
```

```

minix3_init_git_build [Running]
Minix: PID 377 exited
# Minix (200010004): PID 27 swapped in
Minix (200010004): PID 27 swapped in
Minix (200010004): PID 28 swapped in
Minix (200010004): PID 28 swapped in
Minix (200010004): PID 25 swapped in
Minix (200010004): PID 25 swapped in

#
#
#
#
# Minix (200010004): PID 27 swapped in
pwd
/root
# Minix (200010004): PID 27 swapped in
ls
Minix: PID 378 created
Minix (200010004): PID 128 swapped in
.exrc                200010004_lab2_part2.zip  runme1.sh
.profile             byte-unixbench-mod      system.c
200010004_lab2_part1.zip byte-unixbench-mod.zip
Minix: PID 378 exited
# sh runme1.sh

```

Figure 1:

```

minix3_init_git_build [Running]
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
rm /dev/c0d0p0s0:/boot/minix/3.3.0r6
Done.
Build started at: Sun Feb  5 12:04:19 GMT 2023
Build finished at: Sun Feb  5 12:11:13 GMT 2023
Minix: PID 24928 exited
# _

```

Figure 2:

## workload\_mix1.sh

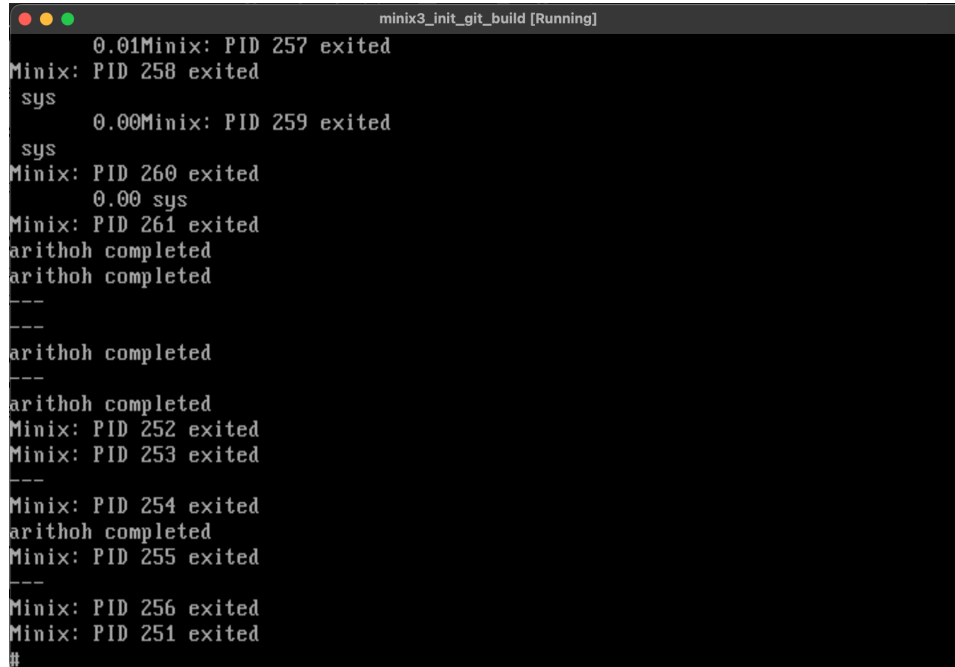
```
#!/bin/sh
./arithoh.sh &
./arithoh.sh &
./arithoh.sh &
./arithoh.sh &
./arithoh.sh &
wait
```

### Observations:

We see that the 5 processes are executed in parallel in the form of shifts. The time account, which is 200 by default, is used in full each time the process is scheduled.

### Inference:

The CPU fairly schedules all processes on a rotational basis. Since these are CPU intensive processes, so they do not need to wait for I/O and run completely in the allotted quanta of time provided to them. These processes used the entire time quantum and placed at the end of the queue.



```
minix3_init_git_build [Running]
0.01Minix: PID 257 exited
Minix: PID 258 exited
sys
0.00Minix: PID 259 exited
sys
Minix: PID 260 exited
0.00 sys
Minix: PID 261 exited
arithoh completed
arithoh completed
---
arithoh completed
---
arithoh completed
Minix: PID 252 exited
Minix: PID 253 exited
---
Minix: PID 254 exited
arithoh completed
Minix: PID 255 exited
---
Minix: PID 256 exited
Minix: PID 251 exited
# _
```

Figure 3:

## workload\_mix2.sh

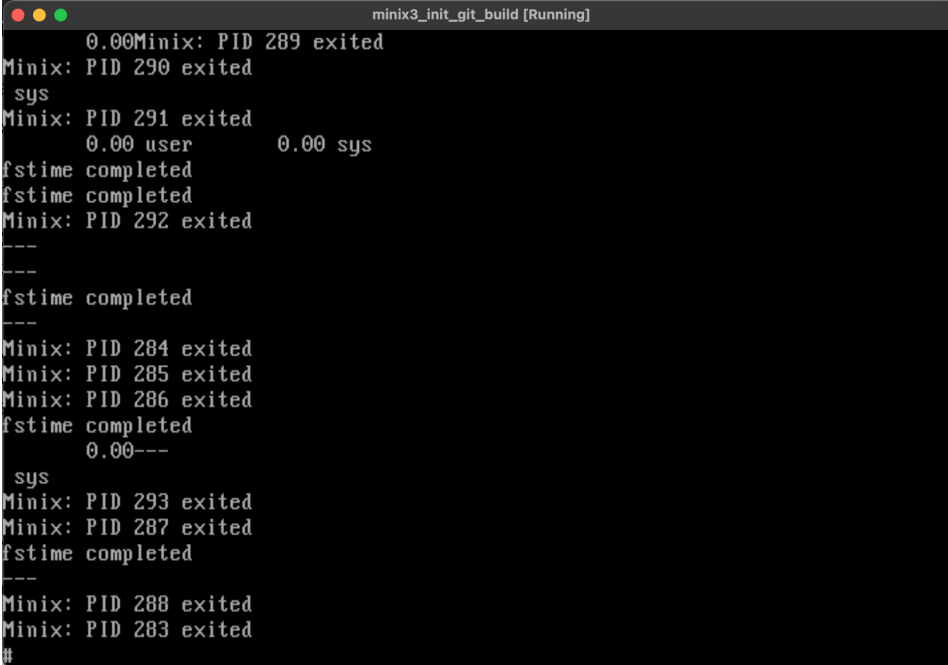
```
#!/bin/sh
./fstime.sh &
./fstime.sh &
./fstime.sh &
./fstime.sh &
./fstime.sh &
wait
```

### Observations:

The 5 processes run in shifts. Also, since modifying files is a system call, it has higher quantum inorder to prevent malfunctioning.

### Inference:

The fstime.sh process is I/O bound. Tasks don't use their quanta to the fullest as they don't consume much CPU as it got blocked waiting for I/O.



```
minix3_init_git_build [Running]
0.00Minix: PID 289 exited
Minix: PID 290 exited
sys
Minix: PID 291 exited
0.00 user    0.00 sys
fstime completed
fstime completed
Minix: PID 292 exited
---
---
fstime completed
---
Minix: PID 284 exited
Minix: PID 285 exited
Minix: PID 286 exited
fstime completed
0.00---
sys
Minix: PID 293 exited
Minix: PID 287 exited
fstime completed
---
Minix: PID 288 exited
Minix: PID 283 exited
#
```

Figure 4:

## workload\_mix3.sh

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

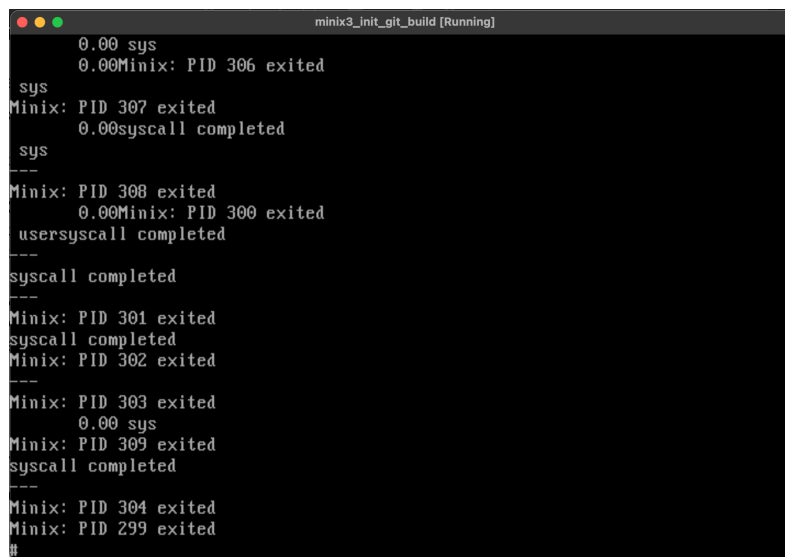
### Observations:

Syscall is a system call that is a request made by a program to the operating system's kernel. A system call can be either CPU bound or I/O bound.

We see that the 5 processes are executed in parallel in the form of shifts. The time account, which is 200 by default, is used in full each time the process is scheduled. Since syscall.sh is not intensive as arithoh.sh, all the quanta slots are not fully engaged every time the process is scheduled.

### Inference:

The CPU fairly schedules all processes on a rotational basis. Since these are lesser CPU intensive processes, and run completely in the allotted quanta of time provided to them. Hence, in this workload mix3.sh, all process finish in almost same time while. partially utilizing their assigned quantum slots.



```
minix3_init_git_build [Running]
0.00 sys
0.00Minix: PID 306 exited
sys
Minix: PID 307 exited
0.00syscall completed
sys
Minix: PID 308 exited
0.00Minix: PID 300 exited
usersyscall completed
syscall completed
Minix: PID 301 exited
syscall completed
Minix: PID 302 exited
Minix: PID 303 exited
0.00 sys
Minix: PID 309 exited
syscall completed
Minix: PID 304 exited
Minix: PID 299 exited
#
```

Figure 5: ■

## workload\_mix4.sh

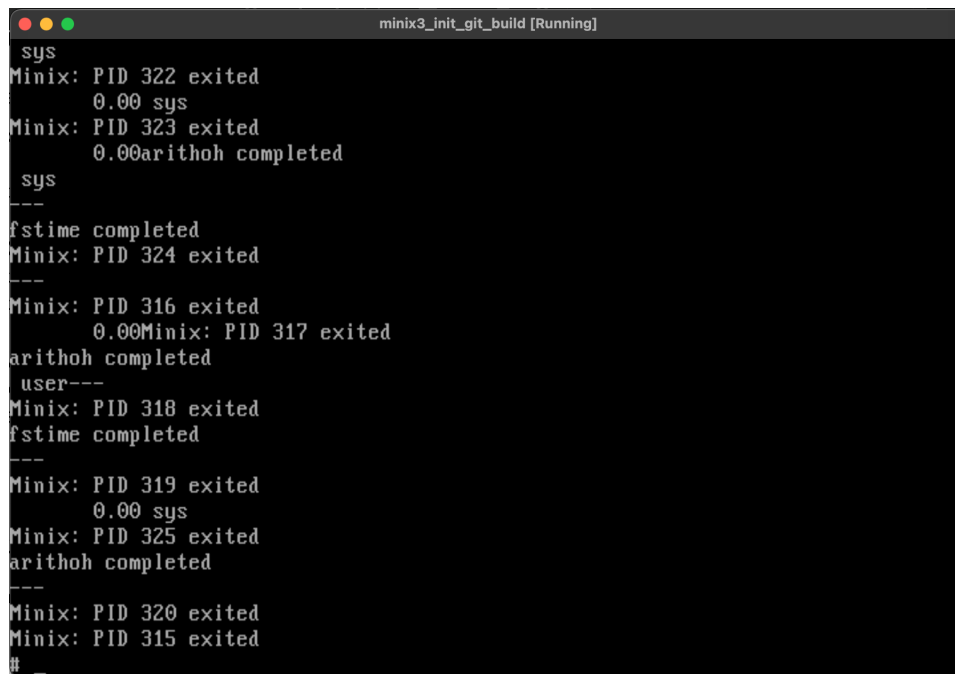
```
#!/bin/sh
./arithoh.sh &
./fstime.sh &
./arithoh.sh &
./fstime.sh &
./arithoh.sh &
wait
```

### Observations:

We see that the 3 CPU intensive processes and the 2 linked I/O processes run in parallel in a round-robin fashion. However, the other 2 linked I/O processes are waiting for their I/O work and are then scheduled to run on the CPU. So after completion these processes are put at the head of the queue with remaining time quantum.

### Inference:

CPU-bound arithoh tasks use all of their quanta, while I/O-bound fstime tasks don't.



```
minix3_init_git_build [Running]
sys
Minix: PID 322 exited
0.00 sys
Minix: PID 323 exited
0.00arithoh completed
sys
---
fstime completed
Minix: PID 324 exited
---
Minix: PID 316 exited
0.00Minix: PID 317 exited
arithoh completed
user---
Minix: PID 318 exited
fstime completed
---
Minix: PID 319 exited
0.00 sys
Minix: PID 325 exited
arithoh completed
---
Minix: PID 320 exited
Minix: PID 315 exited
# _
```

Figure 6:

```
minix3_init_git_build [Running]
# ls
Minix: PID 349 created
Minix (200010004): Allotted Quantum: 200 ms
Minix (200010004): Used Quantum: 200 ms
Minix (200010004): PID 99 swapped in
arithoh.sh      pipe.sh      workload_mix.sh      workload_mix3.sh
fstime.sh       spawn.sh     workload_mix1.sh     workload_mix4.sh
log.txt         syscall.sh   workload_mix2.sh
Minix: PID 349 exited
# rm log.txt
Minix: PID 350 created
Minix (200010004): Allotted Quantum: 200 ms
Minix (200010004): Used Quantum: 200 ms
Minix (200010004): PID 100 swapped in
Minix: PID 350 exited
# ls
Minix: PID 351 created
Minix (200010004): Allotted Quantum: 200 ms
Minix (200010004): Used Quantum: 200 ms
Minix (200010004): PID 101 swapped in
arithoh.sh      spawn.sh     workload_mix1.sh     workload_mix4.sh
fstime.sh       syscall.sh   workload_mix2.sh
pipe.sh         workload_mix.sh   workload_mix3.sh
Minix: PID 351 exited
# ./workload_mix1.sh_
```

Figure 7:

## 2 Part 2

### Problem

Modify the user-level scheduler in Minix3 to the following “Pseudo-FIFO” policy: among the user-level processes that are ready to execute, the one that entered the earliest must be scheduled.

### Answer

The minix 3 scheduler by default following Round Robin policy inside each queue. Whenever a process uses complete quanta of time allocated to it, it will be send the subsequent lower priority queue. To avoid starvation, periodically, all the processes will get the priority boosted after certain interval of time. I/O bound processes run until they block but they are given higher quantum inorder to prevent malfunctioning. If a process has not



used its entire quantum and blocked, then it means it got blocked waiting for i/o. After i/o is complete the process is put at the head of the queue with remaining time quantum. To implement Pseudo-FIFO, we increase the priority of the running process after it completes its time quanta so that the process that was running first has the highest priority and is selected to run again. To do this, we have modified the `do_noquantum()` function of the `schedule.c` file present in the `/minix/servers/sched/` directory as follows:

```
rmp->priority -= 1; /* higher priority */
```

Next, to avoid overflowing of queue, we have to ensure periodic priority boosting doesn't occur so that user processes don't invade the queues of drivers, servers, clock and system task. To do this we have commented out the following code present in the `balance_queues()` function of the same file.

```
// rmp->priority -= 1; /* increase priority */
```

To copy the modified file and re-build the OS, we have written a bash file named `runme2.sh`. Upon executing `runme2.sh`, and successful build, followed by reboot, the changes should be reflected. After this, we ran the workloads of part 1 again and our observations and inferences are presented in sections below.

```
#!/bin/bash

cp schedule.c /usr/src/minix/servers/sched/schedule.c;
cd ../usr/src;
make build MKUPDATE=yes > log.txt;
```

```

minix3_init_git_build [Running]
Minix: PID 377 exited
# Minix (200010004): PID 27 swapped in
Minix (200010004): PID 27 swapped in
Minix (200010004): PID 28 swapped in
Minix (200010004): PID 28 swapped in
Minix (200010004): PID 25 swapped in
Minix (200010004): PID 25 swapped in

#
#
#
#
#
# Minix (200010004): PID 27 swapped in
pwd
/root
# Minix (200010004): PID 27 swapped in
ls
Minix: PID 378 created
Minix (200010004): PID 128 swapped in
.exrc                200010004_lab2_part2.zip  runme1.sh
.profile             byte-unixbench-mod      system.c
200010004_lab2_part1.zip byte-unixbench-mod.zip
Minix: PID 378 exited
# sh runme1.sh

```

Figure 8:

```

minix3_init_git_build [Running]
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
rm /dev/c0d0p0s0:/boot/minix/3.3.0r6
Done.
Build started at: Sun Feb  5 12:04:19 GMT 2023
Build finished at: Sun Feb  5 12:11:13 GMT 2023
Minix: PID 24928 exited
# _

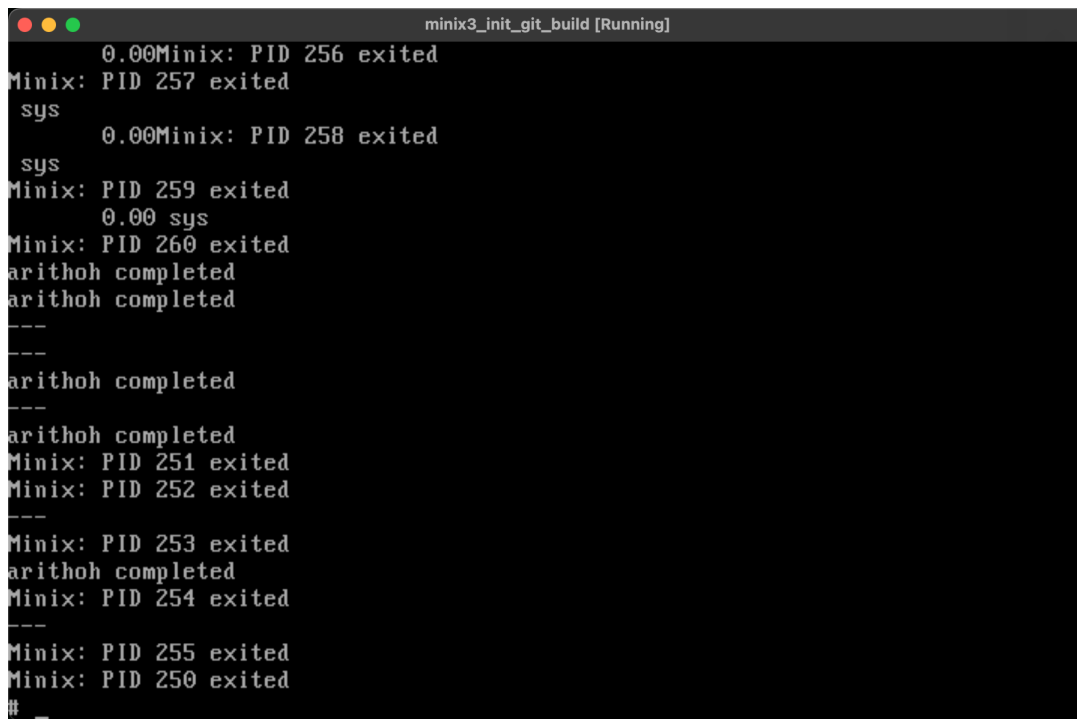
```

Figure 9:

## workload\_mix1.sh

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

This workload spawns 5 different instances of arithoh.sh process, which is CPU bound. Since all 5 are CPU Intensive processes, they run one after the other sequentially in first come first serve fashion. In the execution, they fully engage their quantum slot of 200 as they do not have to wait for I/O. All of these processes were allocated a time quanta of 200 ms by default. Since they are CPU bound, they utilize the complete time quanta. Unlike round robin manner, in the Pseudo-FIFO, one process starts its execution and runs till completion. In the figure 10, we can see process with same PID , is being scheduled back to back indicating a FIFO policy.



```
minix3_init_git_build [Running]
0.00Minix: PID 256 exited
Minix: PID 257 exited
sys
0.00Minix: PID 258 exited
sys
Minix: PID 259 exited
0.00 sys
Minix: PID 260 exited
arithoh completed
arithoh completed
---
arithoh completed
---
arithoh completed
Minix: PID 251 exited
Minix: PID 252 exited
---
Minix: PID 253 exited
arithoh completed
Minix: PID 254 exited
---
Minix: PID 255 exited
Minix: PID 250 exited
# _
```

Figure 10:

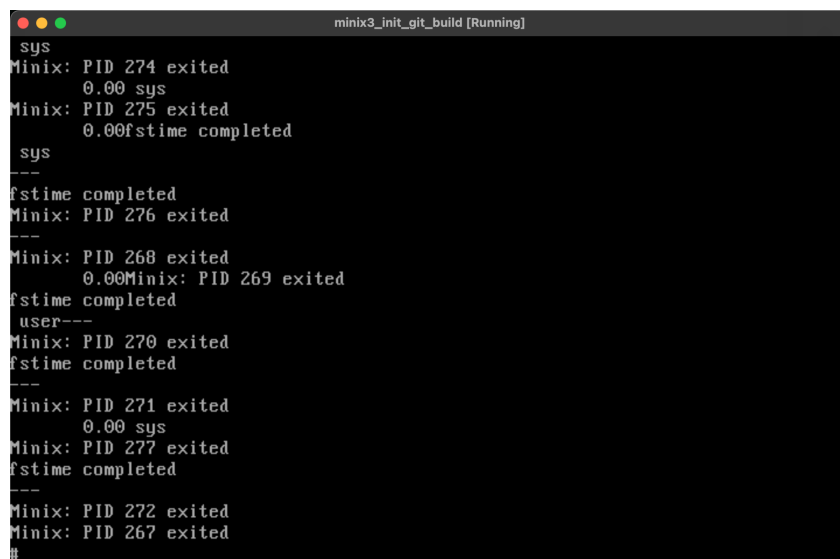
## workload\_mix2.sh

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

This workload spawns 5 different instances of `fstime.sh` process, which is I/O bound.

Again in this workload we can observe the pseudo nature of the Pseudo FIFO policy. The `fstime.sh` is I/O bound process. By default these processes were allocated a time quanta of 200 ms, but don't completely utilize it. So next time they are allocated a time quanta of 500 ms. Unlike an actual FIFO policy, in our case when ever a process waits for I/O to respond, we simply preempt it. This was also observed in the case of round robin policy of default minix. Once the I/O responds, the processes are scheduled again until completion in a FIFO manner assuming they do not depend on I/O anymore. This is the reason we call our policy "Pseudo" FIFO. This policy follows FIFO, only in the case of CPU bound processes or when there is no dependency on I/O devices.

Since the I/O bound processes are sent to the waiting queue after requesting for I/O and are then placed back in the ready queue and scheduled to work on the CPU when I/O received. I/O bound tasks don't always utilize complete quanta slot of 500 allotted to them. Hence they then follow normal Round-Robin Order itself. There is the no change observed when Pseudo FIFO is applied rather than default Round-Robin in Minix3 in this case



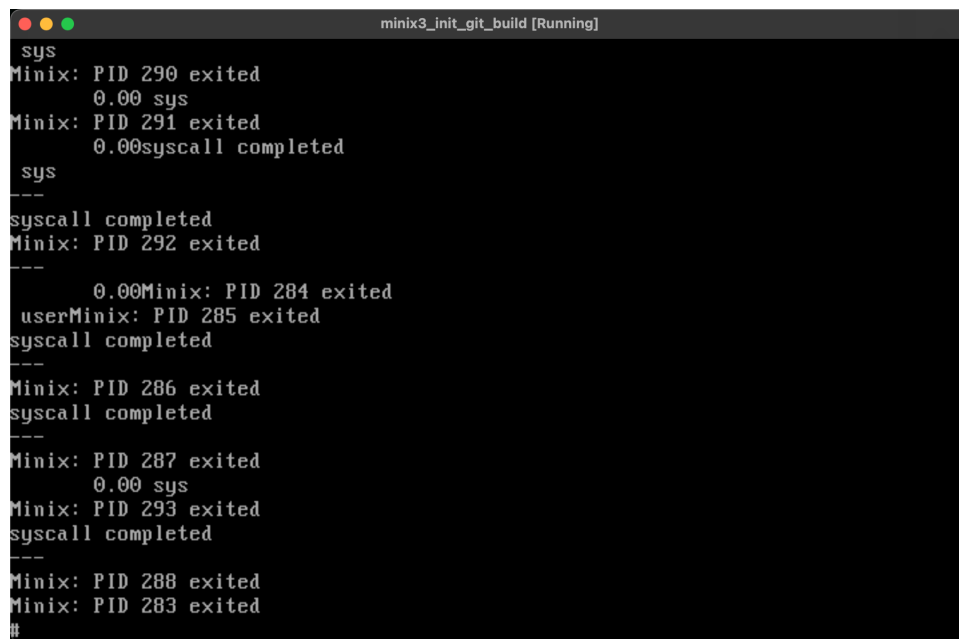
```
minix3_init_git_build [Running]
sys
Minix: PID 274 exited
0.00 sys
Minix: PID 275 exited
0.00fstime completed
sys
---
fstime completed
Minix: PID 276 exited
---
Minix: PID 268 exited
0.00Minix: PID 269 exited
fstime completed
user---
Minix: PID 270 exited
fstime completed
---
Minix: PID 271 exited
0.00 sys
Minix: PID 277 exited
fstime completed
---
Minix: PID 272 exited
Minix: PID 267 exited
#
```

Figure 11:

## workload\_mix3.sh

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

The syscall.sh is CPU bound process. By default these processes have been allocated a time quanta of 200 ms. These processes run one after the other in a FIFO manner, unlike round robin manner of default minix. During such execution, processes fully engage their quantum slots of 200 as they do not wait for I/O. Only when one process's execution is terminated, next process is scheduled subsequently. From figure 12, we can see that process with same PID is being scheduled back to back until completion. Unlike round robin, here once a process completes its execution, only then the next process is scheduled.



```
minix3_init_git_build [Running]
sys
Minix: PID 290 exited
0.00 sys
Minix: PID 291 exited
0.00syscall completed
sys
---
syscall completed
Minix: PID 292 exited
---
0.00Minix: PID 284 exited
userMinix: PID 285 exited
syscall completed
---
Minix: PID 286 exited
syscall completed
---
Minix: PID 287 exited
0.00 sys
Minix: PID 293 exited
syscall completed
---
Minix: PID 288 exited
Minix: PID 283 exited
#
```

Figure 12:

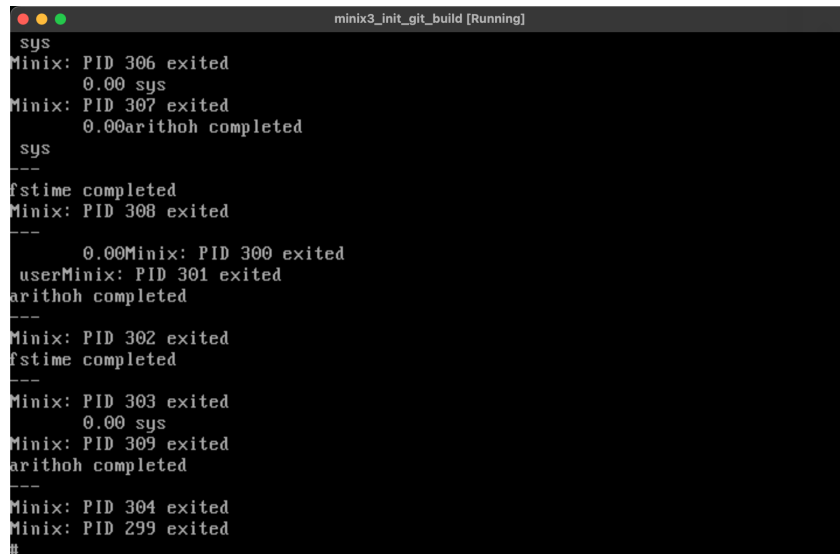
## workload\_mix4.sh

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

This workload spawns 3 different instances of arithoh.sh process, which is CPU bound and 2 different instances of fstime.sh process which is I/O bound.

By default, all of these process have been allocated a time quanta of 200 ms. In this case, we can observe the working of actual "Pseudo" FIFO. In an actual FIFO policy, when an I/O bound process is scheduled, the CPU has to wait until I/O responds, while the I/O bound fstime.sh processes wait for I/O to respond, the scheduler schedules CPU bound arithoh.sh processes. But unlike the default minix scheduler, this scheduler follows a FIFO policy in case of CPU bound processes. As seen from figure 13, process with same PID is scheduled back to back indicating a FIFO policy. The fstime.sh processes being I/O bound do not completely utilize the allocated time quanta and they are allocated a time quanta of 500 ms next time. After the I/O responds, the fstime.sh processes are scheduled and completed.

So, we can conclude that after our appropriate changes in codes, CPU intensive were able to follow proper FIFO scheduling whereas I/O bound processes weren't. Due to this approximation and exceptions, we declare such scheduling as Pseudo FIFO.



```
minix3_init_git_build [Running]
sys
Minix: PID 306 exited
0.00 sys
Minix: PID 307 exited
0.00arithoh completed
sys
---
fstime completed
Minix: PID 308 exited
---
0.00Minix: PID 300 exited
userMinix: PID 301 exited
arithoh completed
---
Minix: PID 302 exited
fstime completed
---
Minix: PID 303 exited
0.00 sys
Minix: PID 309 exited
arithoh completed
---
Minix: PID 304 exited
Minix: PID 299 exited
#
```

Figure 13: