Operating Systems Lab - CS 314

Rishit Saiya - 180010027, Assignment - 4

February 12, 2021

1 Part-1

In this part, we had to prepare 4 different workload mixes having varying characteristics, ranging from all compute-intensive benchmarks to all I/O to CPU Intensive benchmarks. Furthermore, each workload should spawn around 5 processes.

Before doing analysis on UnixBench, the configuration files to display quanta had to be changed. 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("Allotted Quantum is: %d, Used Quantum is: %d\n",
p->p_quantum_size_ms,
p->p_quantum_size_ms-cpu_time_2_ms(p->p_cpu_time_left));
```

This was sent to Minix3 for a build using the following script `run1.sh`:

```
cp system.c /usr/src/minix/kernel/;
cd /usr/src/;
make build MKUPDATE=yes >log.txt 2>log.txt
```

Using the given reference, in problem statement, further analysis on the time quanta spent in the CPU by the processes were made and the analysis are as follows:

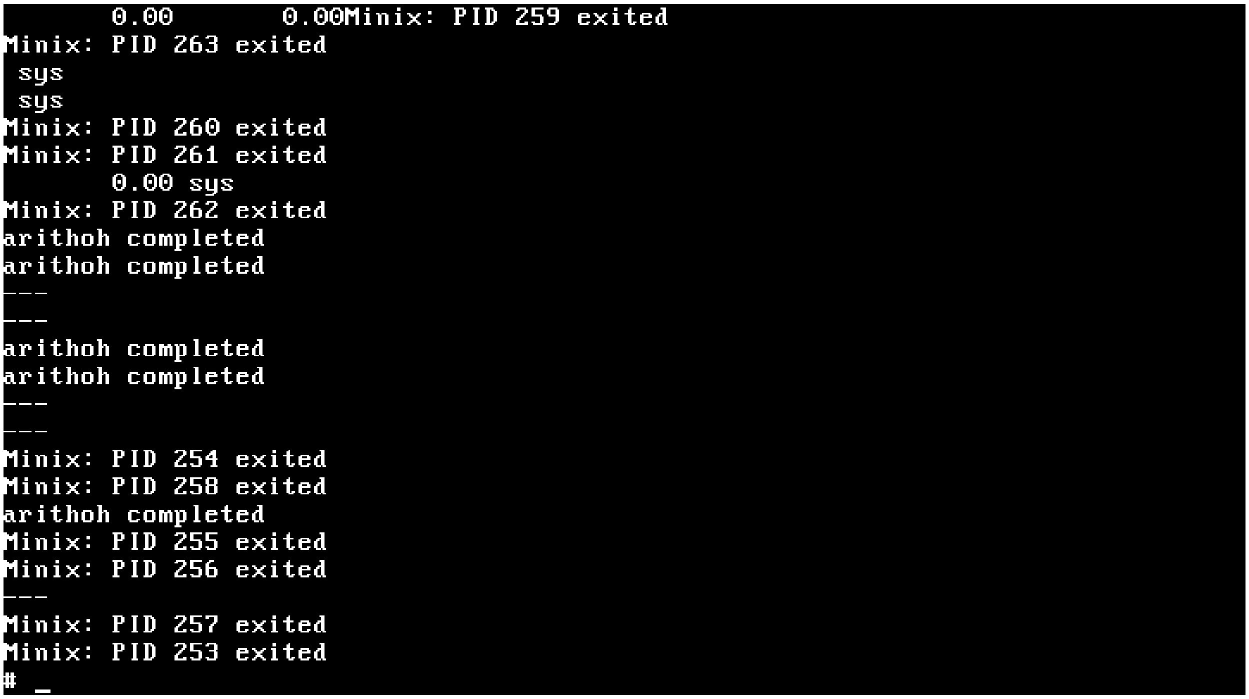
1.1 workload_mix1.sh

In `workload_mix1.sh`, the following script was added:

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

In the above script, all the 5 processes were of `arithoh.sh` with PIDs.

Observations & Inference: All these 5 processes execute parallelly in Round-Robin fashion which is default fashion in Minix3. The time quanta assigned to such CPU Intensive tasks by Minix3 is 200 by default. During such executions, the 200 quanta allocation is completely engaged every time the process is scheduled. From literature, we know that CPU Intensive processes do not need I/O and hence they execute fully in the time quanta they are allocated. Hence, in this `workload_mix1.sh`, all process finish in almost same time while fully utilizing their assigned quantum slots.



```
       0.00       0.00Minix: PID 259 exited
Minix: PID 263 exited
 sys
 sys
Minix: PID 260 exited
Minix: PID 261 exited
       0.00 sys
Minix: PID 262 exited
arithoh completed
arithoh completed
---
---
arithoh completed
arithoh completed
---
---
Minix: PID 254 exited
Minix: PID 258 exited
arithoh completed
Minix: PID 255 exited
Minix: PID 256 exited
---
Minix: PID 257 exited
Minix: PID 253 exited
# _
```

Figure 1: workload_mix1.sh

1.2 workload_mix2.sh

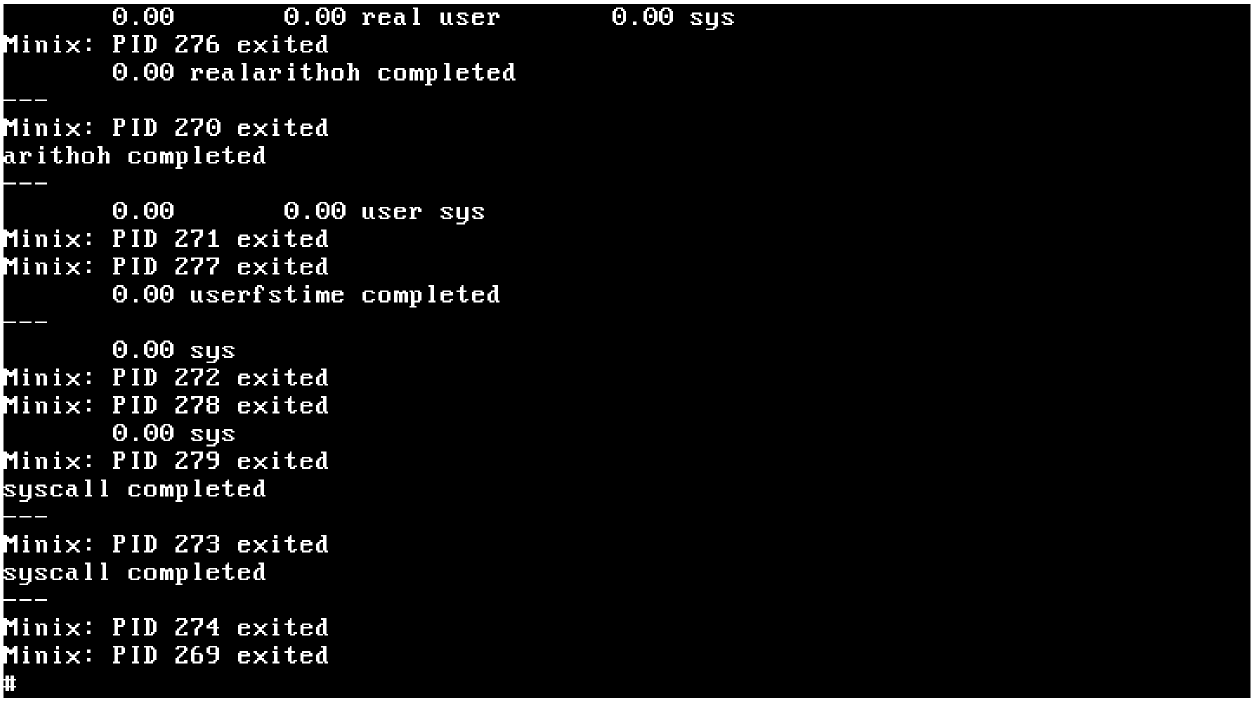
In `workload_mix2.sh`, the following script was added:

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

This script of `workload_mix2.sh` consists of `arithoh.sh`, `fstime.sh` & `syscall.sh`

Observations & Inference: `arithoh.sh` is CPU Intensive, `fstime.sh` is I/O Bound and `syscall.sh` is System CPU Intensive. So, with different PIDs PID 269, 270: `arithoh.sh`, PID 271: `fstime.sh`, PID 273, 279: `syscall.sh`.

Clearly, CPU Intensive Tasks execute in a Round Robin fashion, whilst I/O bound processes wait for I/O for execution. Since, `arithoh.sh` is a CPU Intensive Process which completely utilizes its allotted quanta and finishes its task and exits. Next, `fstime.sh` allotted more quanta (500) than a CPU Intensive task (200) and I/O Bound process doesn't utilize all quanta every time. After all other processes are completed and then both `arithoh.sh` are scheduled in Round Robin fashion until they are completed. So, more CPU Intensive processes take more time where as I/O bound processes wait for I/O to be completed.



```
       0.00       0.00 real user       0.00 sys
Minix: PID 276 exited
       0.00 realarithoh completed
---
Minix: PID 270 exited
arithoh completed
---
       0.00       0.00 user sys
Minix: PID 271 exited
Minix: PID 277 exited
       0.00 userfstime completed
---
       0.00 sys
Minix: PID 272 exited
Minix: PID 278 exited
       0.00 sys
Minix: PID 279 exited
syscall completed
---
Minix: PID 273 exited
syscall completed
---
Minix: PID 274 exited
Minix: PID 269 exited
#
```

Figure 2: workload_mix2.sh

1.3 workload_mix3.sh

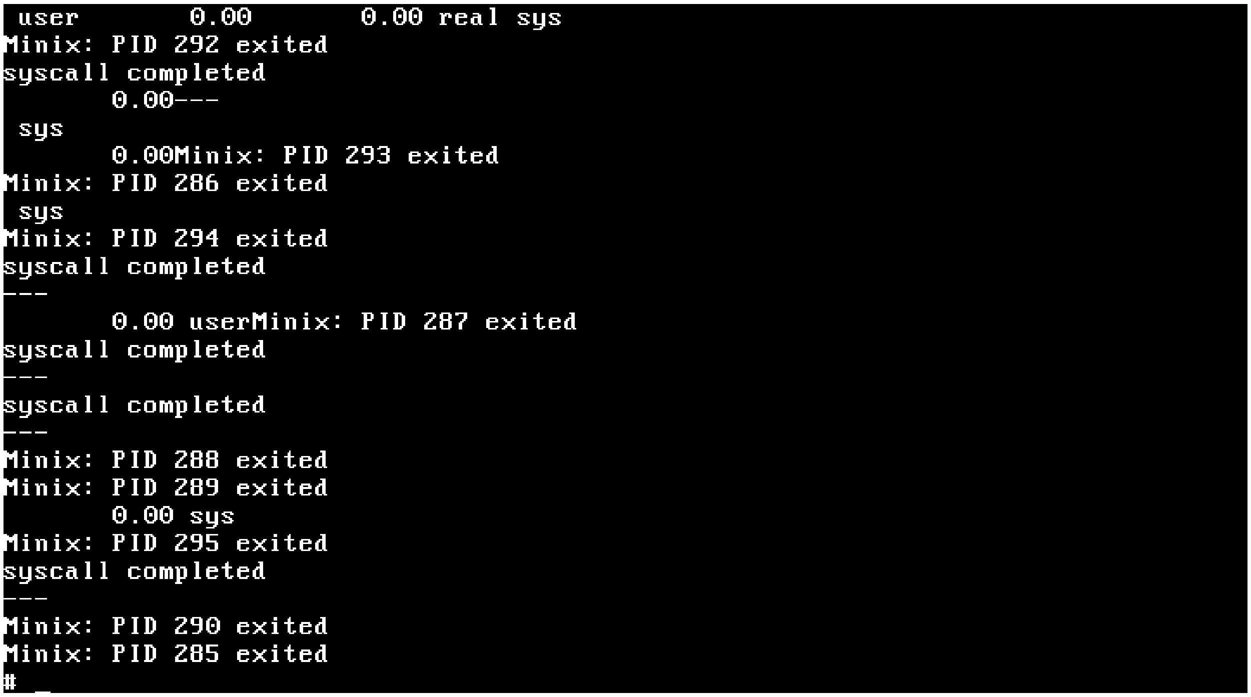
In `workload_mix3.sh`, the following script was added:

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

```
./syscall.sh &
./syscall.sh &
wait
```

In the above script, all the 5 processes were of `syscall.sh` with PIDs.

Observations & Inference: All these 5 processes execute parallelly in Round-Robin fashion which is default fashion in Minix3. The time quanta as mentioned above, assigned to such CPU Intensive task is 200. Since `syscall.sh` is not intensive as `arithoh.sh`, all the quanta slots Are not fully engaged every time the process is scheduled. CPU fairly schedules all the processes in a round-robin fashion. From literature, we know that CPU Intensive processes do not need I/O and hence they execute fully in the time quanta they are allocated. Hence, in this `workload_mix3.sh`, all process finish in almost same time while partially utilizing their assigned quantum slots.



```
 user       0.00       0.00 real sys
Minix: PID 292 exited
syscall completed
       0.00---
 sys
       0.00Minix: PID 293 exited
Minix: PID 286 exited
 sys
Minix: PID 294 exited
syscall completed
---
       0.00 userMinix: PID 287 exited
syscall completed
---
syscall completed
---
Minix: PID 288 exited
Minix: PID 289 exited
       0.00 sys
Minix: PID 295 exited
syscall completed
---
Minix: PID 290 exited
Minix: PID 285 exited
# _
```

Figure 3: workload_mix3.sh

1.4 workload_mix4.sh

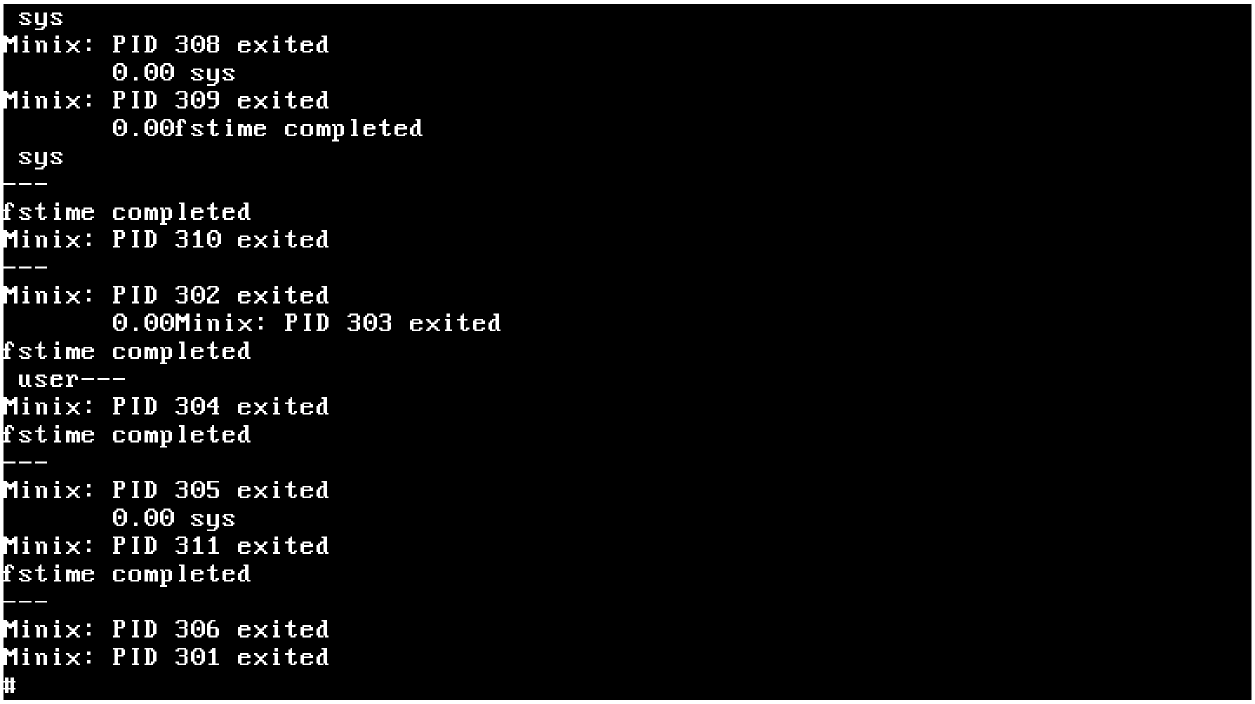
In `workload_mix4.sh`, the following script was added:

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

```
./fstime.sh &
./fstime.sh &
./fstime.sh &
wait
```

In the above script, all the 5 processes were of `fstime.sh` with PIDs.

Observations & Inference: `fstime.sh` is intrinsically I/O bound in nature. Hence in this `workload_mix4.sh` all the process would execute sequentially with their scripts in Round-Robin Fashion. Since we have 5 I/O bound processes wait for their I/O work and then are scheduled to work on the CPU. By literature, I/O bound processes do not utilize all allotted quanta (500). All processes wait until they receive I/O and then all complete their operations in that order.



```
 sys
Minix: PID 308 exited
       0.00 sys
Minix: PID 309 exited
       0.00fstime completed
 sys
---
fstime completed
Minix: PID 310 exited
---
Minix: PID 302 exited
       0.00Minix: PID 303 exited
fstime completed
 user---
Minix: PID 304 exited
fstime completed
---
Minix: PID 305 exited
       0.00 sys
Minix: PID 311 exited
fstime completed
---
Minix: PID 306 exited
Minix: PID 301 exited
#
```

Figure 4: workload_mix4.sh

2 Part-2

In this part, we had to 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. Minix3 by default follows Round Robin Scheduling and was changed to Pseudo FIFO by following modifications.

Before doing analysis on UnixBench, the configuration files to display quanta had to be changed. For that, file `schedule.c` at location `minix/servers/sched/` was changed slightly. In the function `sched_proc()`, the quanta allotted and used were extracted as follows:

In the function `do_noquantum()`, following changes were made to prioritize in the queue:

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

Now in order to balance the increase priority, we need to also make sure the queue length is not overflowed. For that, we can check the `balance_queues` function and change the following lines:

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

Basically, we want to comment the decrement here or else the priority queue will be overflowed.

2.1 workload_mix1.sh

In `workload_mix1.sh`, the following script was added:

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

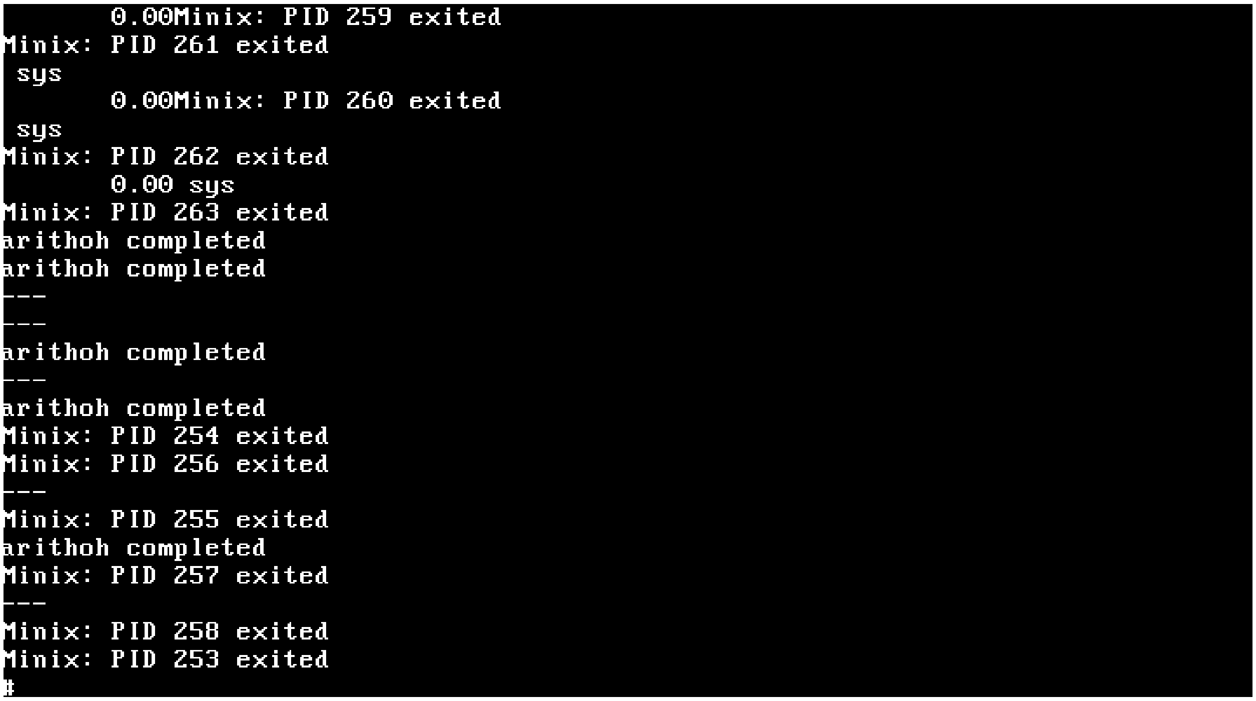
In the above script, all the 5 processes were of `arithoh.sh` with PIDs.

Observations & Inference: 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. This is change observed when Pseudo FIFO is applied rather than default Round-Robin in Minix3.

2.2 workload_mix2.sh

In `workload_mix2.sh`, the following script was added:

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



```
       0.00Minix: PID 259 exited
Minix: PID 261 exited
 sys
       0.00Minix: PID 260 exited
 sys
Minix: PID 262 exited
       0.00 sys
Minix: PID 263 exited
arithoh completed
arithoh completed
---
---
arithoh completed
---
arithoh completed
Minix: PID 254 exited
Minix: PID 256 exited
---
Minix: PID 255 exited
arithoh completed
Minix: PID 257 exited
---
Minix: PID 258 exited
Minix: PID 253 exited
#
```

Figure 5: workload_mix1.sh

This script of `workload_mix2.sh` consists of `arithoh.sh`, `fstime.sh` & `syscall.sh`

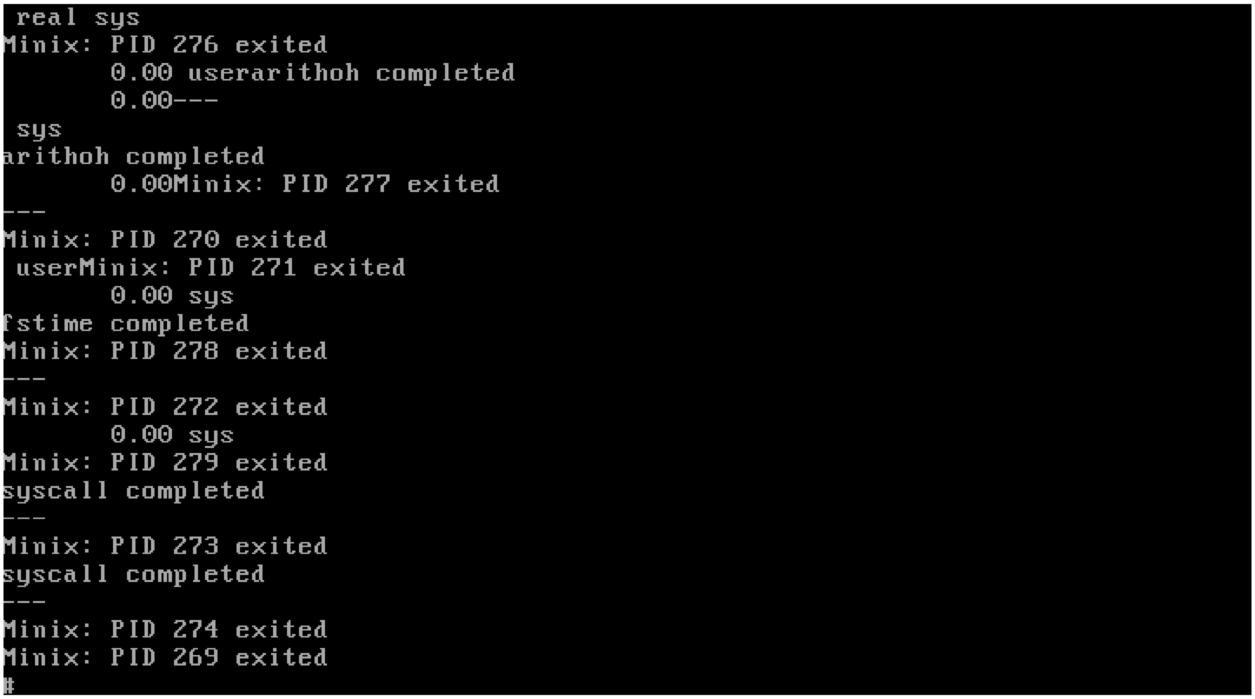
Observations & Inference: In the above script, `arithoh.sh` is repetitively scheduled till its execution is done as the scheduling is in Pseudo FIFO fashion. Both `arithoh.sh` are consecutively scheduled. In the next timeline, `syscall.sh` is completed before `fstime.sh` because I/O bound processes are 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. Thus, `syscall.sh` is completed before `fstime.sh`. Later on, `fstime.sh` is completed at the end when I/O received.

In this execution, `fstime.sh` came before `syscall.sh` so in FIFO order, former was to be completed before latter but however, this is due to the fact that 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.

2.3 workload_mix3.sh

In `workload_mix3.sh`, the following script was added:

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



```
 real sys
Minix: PID 276 exited
       0.00 userarithoh completed
       0.00---
 sys
arithoh completed
       0.00Minix: PID 277 exited
---
Minix: PID 270 exited
 userMinix: PID 271 exited
       0.00 sys
fstime completed
Minix: PID 278 exited
---
Minix: PID 272 exited
       0.00 sys
Minix: PID 279 exited
syscall completed
---
Minix: PID 273 exited
syscall completed
---
Minix: PID 274 exited
Minix: PID 269 exited
#
```

Figure 6: workload_mix2.sh

```
./syscall.sh &
./syscall.sh &
wait
```

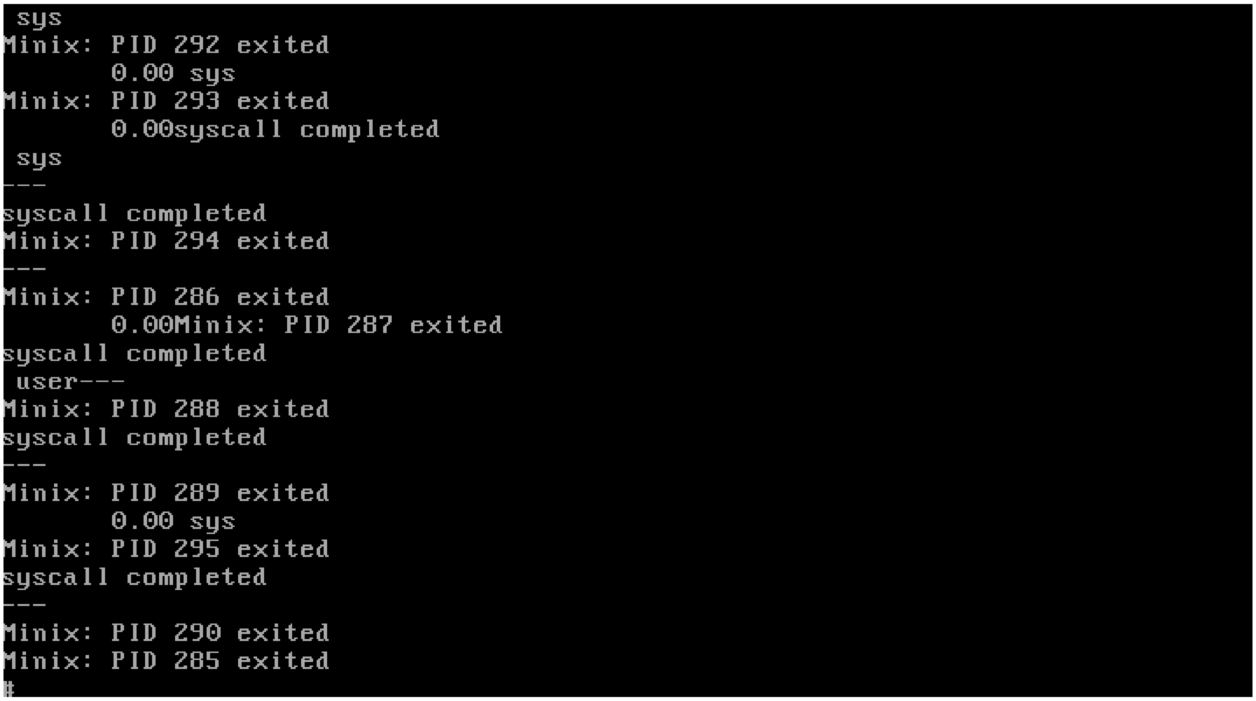
In the above script, all the 5 processes were of `syscall.sh` with PIDs.

Observations & Inference: Since all 5 processes are CPU intensive processes, but less intensive than `arithoh.sh`, we can prioritize them. Here, all 5 CPU Intensive processes run one after the other in first come first serve fashion. 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. This is change observed when Pseudo FIFO is applied rather than default Round-Robin in Minix3.

2.4 workload_mix4.sh

In `workload_mix4.sh`, the following script was added:

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



```
 sys
Minix: PID 292 exited
       0.00 sys
Minix: PID 293 exited
       0.00syscall completed
 sys
---
syscall completed
Minix: PID 294 exited
---
Minix: PID 286 exited
       0.00Minix: PID 287 exited
syscall completed
 user---
Minix: PID 288 exited
syscall completed
---
Minix: PID 289 exited
       0.00 sys
Minix: PID 295 exited
syscall completed
---
Minix: PID 290 exited
Minix: PID 285 exited
#
```

Figure 7: workload_mix3.sh

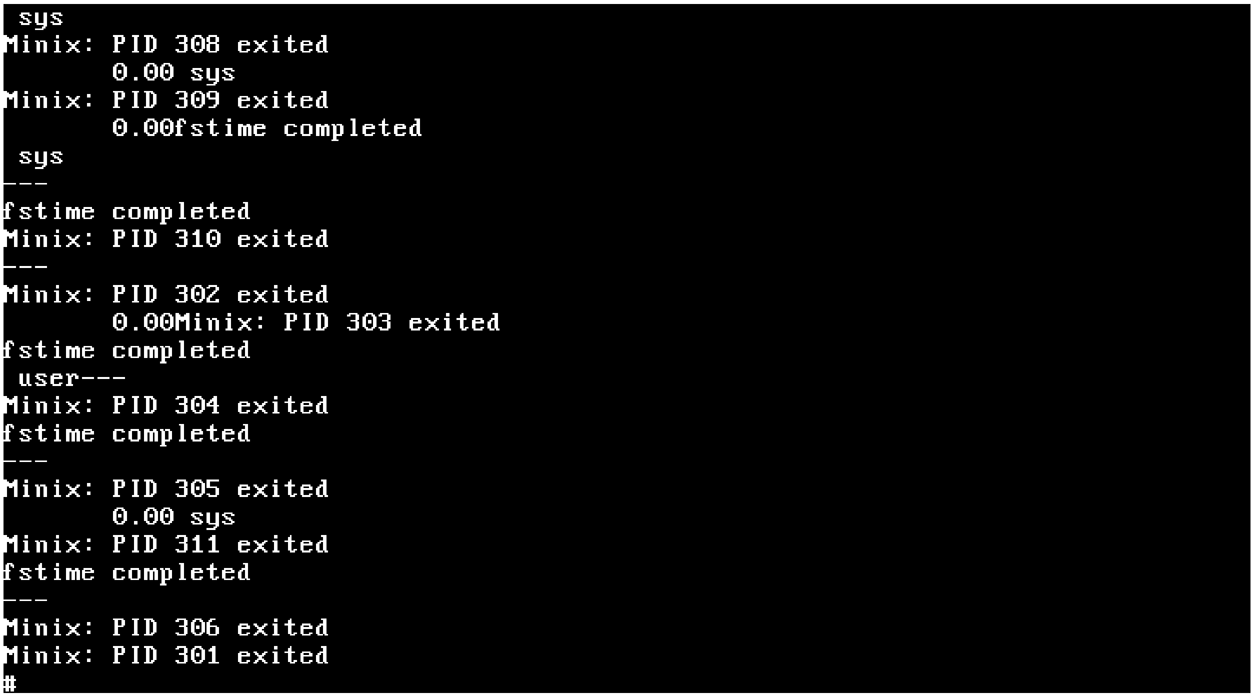
```
./fstime.sh &
wait
```

In the above script, all the 5 processes were of `fstime.sh` with PIDs.

Observations & Inference: `fstime.sh` is intrinsically I/O bound in nature. It is clear from output that here Pseudo FIFO order is not followed accurately in I/O bound processes. This is the exact point why this implementation is called Pseudo and such cases can be exceptions eventually leading this process to be called as Approximation to FIFO Implementation or Pseudo FIFO Implementation.

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. This is the no change observed when Pseudo FIFO is applied rather than default Round-Robin in Minix3 in this case

So, finally we could see 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.



```
 sys
Minix: PID 308 exited
       0.00 sys
Minix: PID 309 exited
       0.00fstime completed
 sys
---
fstime completed
Minix: PID 310 exited
---
Minix: PID 302 exited
       0.00Minix: PID 303 exited
fstime completed
 user---
Minix: PID 304 exited
fstime completed
---
Minix: PID 305 exited
       0.00 sys
Minix: PID 311 exited
fstime completed
---
Minix: PID 306 exited
Minix: PID 301 exited
#
```

Figure 8: workload_mix4.sh