Assignment-2
CS-314:Operating Systems Laboratory

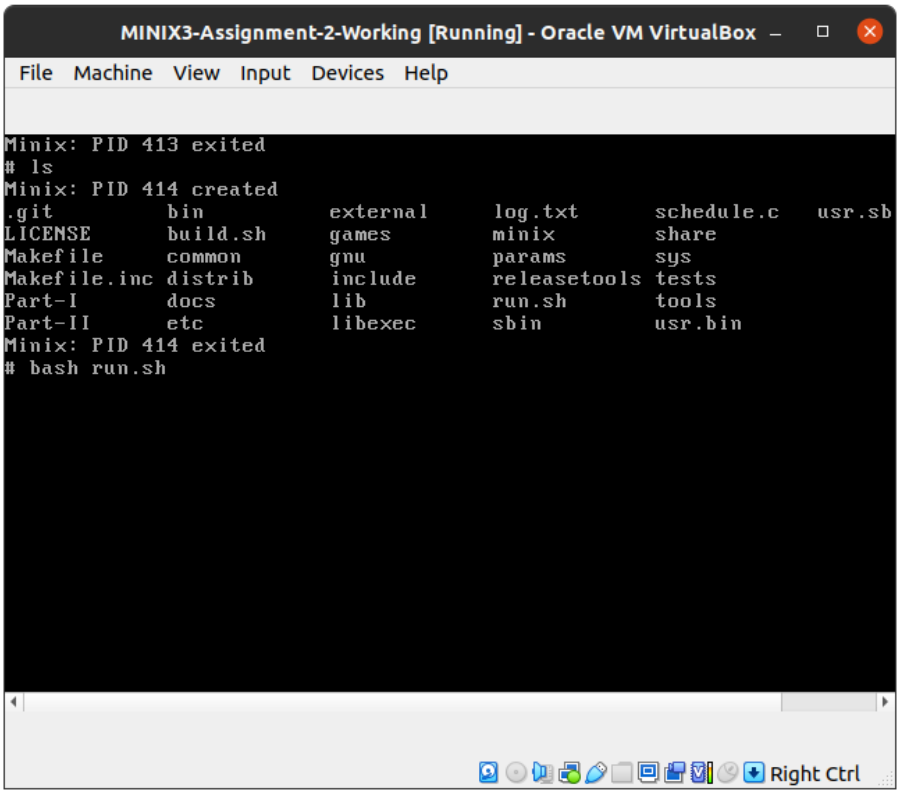
Shahil Patel-200010039
Pranav Talegaonkar-200010041

Part-1

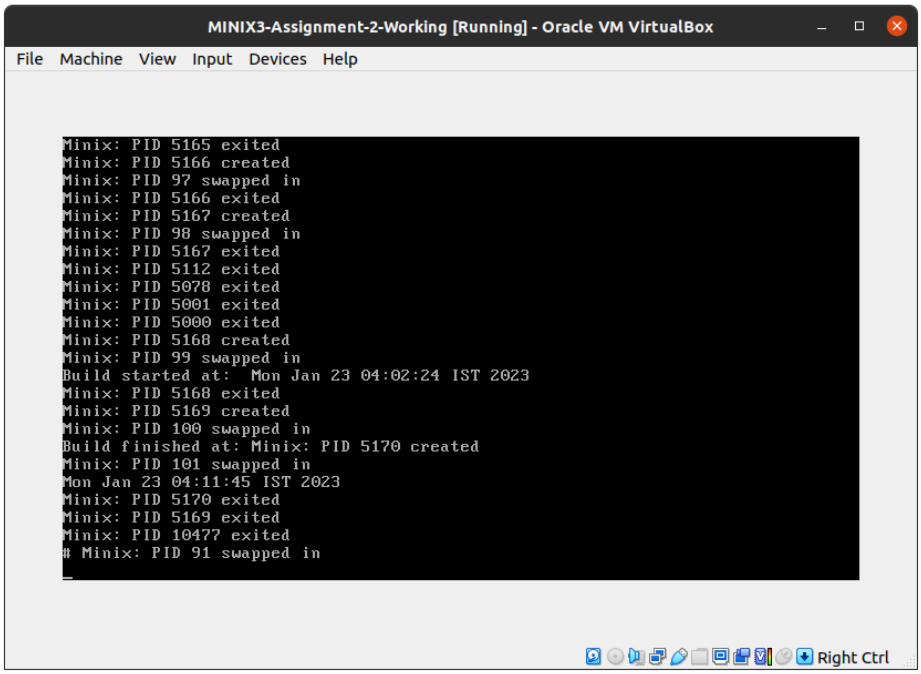
In part 1, we have updated the *Minix-3* source code so that anytime a user-level process is brought in by the scheduler, the string "*PID <pid> swapped in"* is printed.
To accomplish this, we added the following lines of C code to the *schedule.c* file, which can be found in the */minix/minix/servers/sched* directory.

```
if(rmp->priority >= USER_Q){
      printf("(200010039, 200010041)Minix: PID %d swapped in\n",
_ENDPOINT_P(rmp->endpoint));
}
```

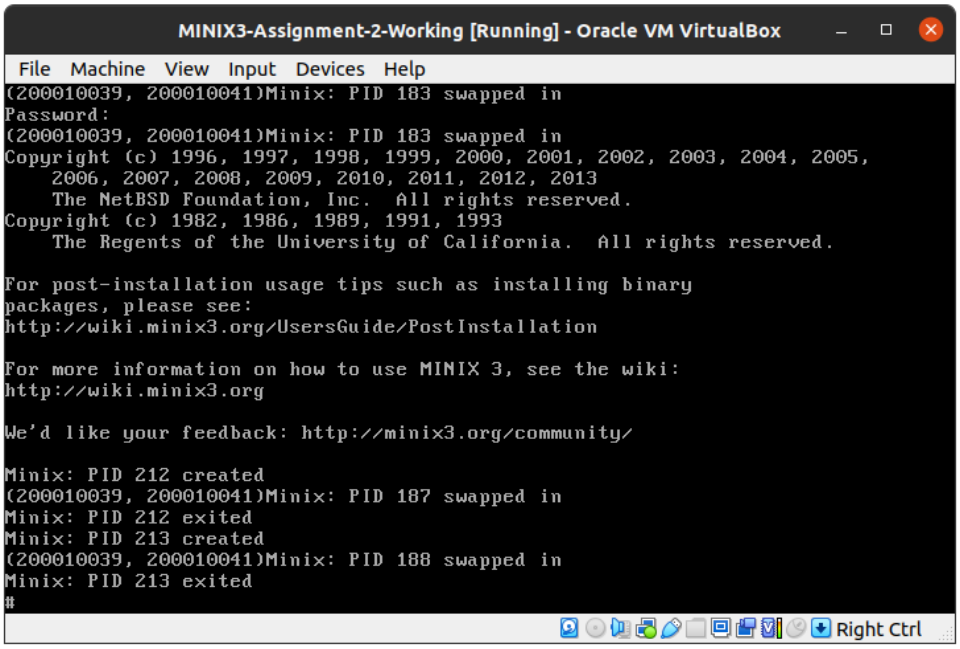
Also to make the appropriate changes to the Minix machine we have made ***run.sh*** which will copy the modified code to appropriate location in the Minix machine and rebuild the Minix OS. The changes are reflected after a successful build and reboot.



Copied files and command to run the run.sh file



Successful build



Changes after reboot

Part-2

In this section, we have used the **UnixBench** benchmark suite to test various work-load combinations in order to understand and analyse the minix scheduler and scheduling orders. The sections that follow discuss the various workloads that are used.

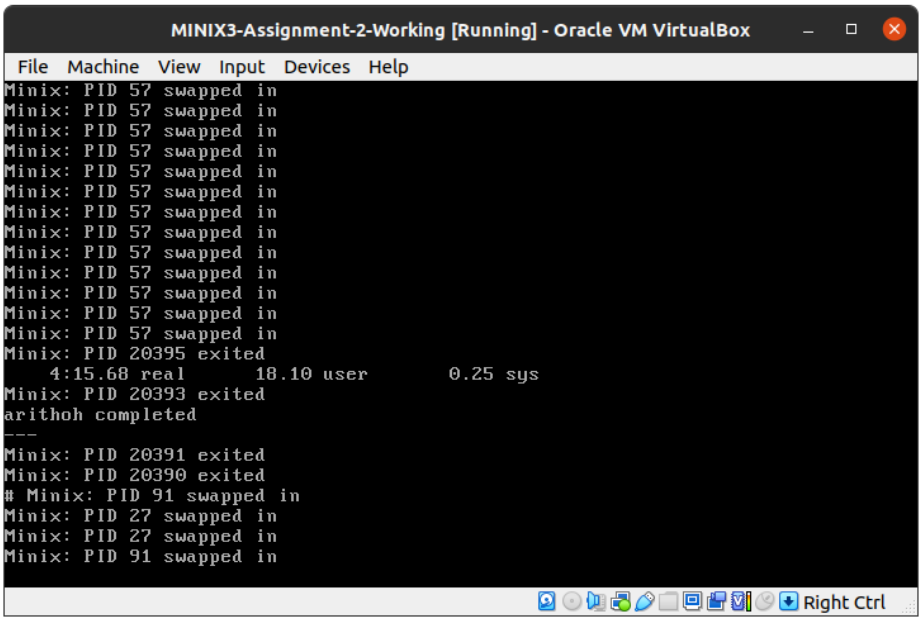
a. workload_mix1.sh

The code for the workload_mix1.sh is as follows:

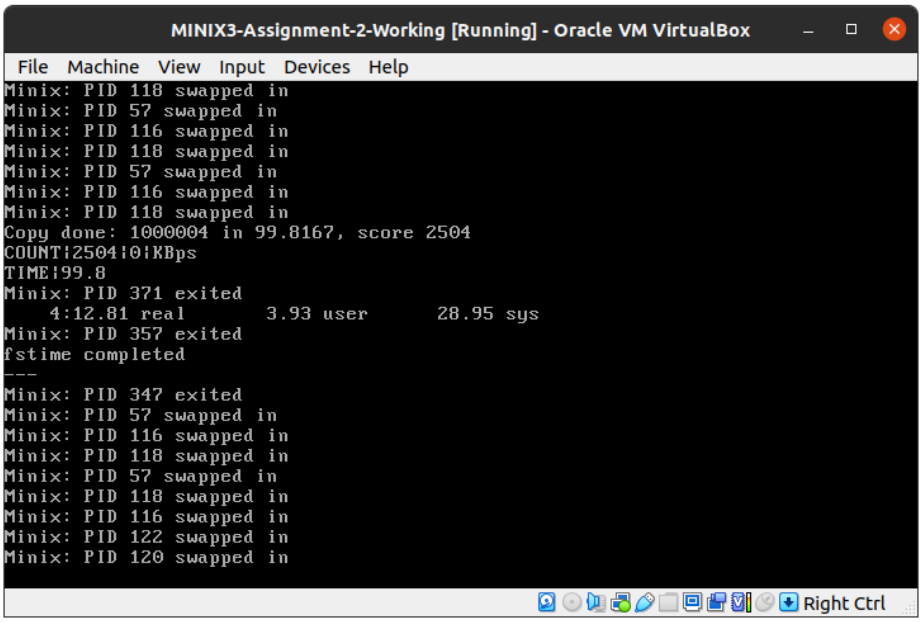
```
#!/bin/sh
./arithoh.sh &
./fstime.sh &
wait
```

In this workload we have used two workloads, namely ***arithoh.sh*** and ***fstime.sh***.

After inspecting their source code and scheduling order, we discovered that *arithoh.sh* is *CPU bound* and does certain CPU bound arithmetic calculations, whereas *fstime.sh* is *I/O bound* and reads and writes some buffer data into memory.



arithoh.sh completed



fstime.sh completed

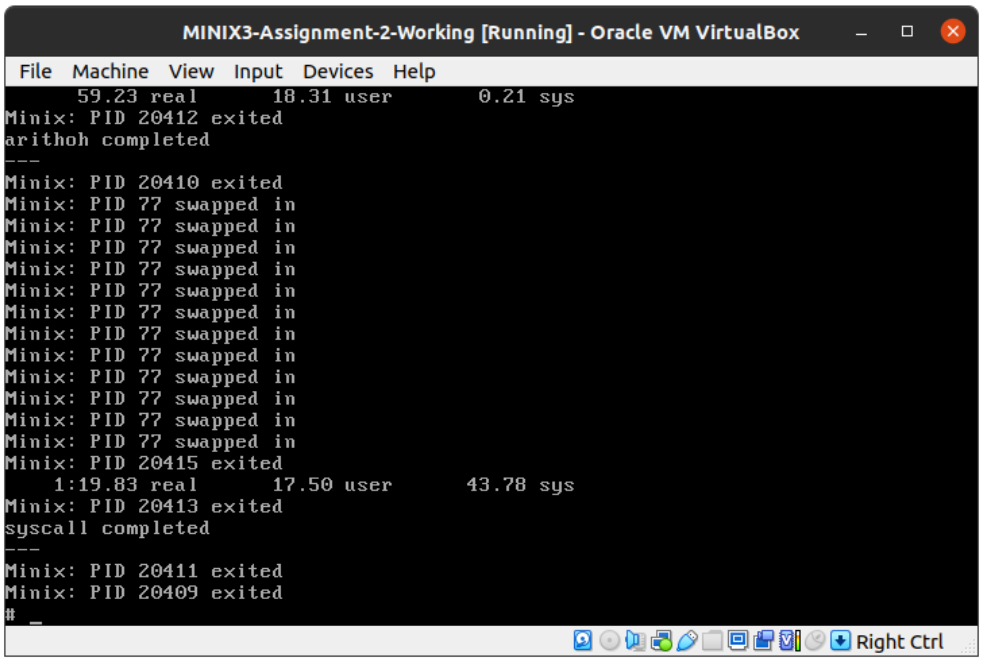
From the above figure we can see that arithoh.sh, with PID 57, being swapped in while fstime.sh, with PID 118 is waiting for I/O response. When fstime.sh becomes ready after I/O response, it is scheduled since it has relatively smaller burst time and it is completed as seen in the above figure After this the arithoh.sh is scheduled until completion as seen from figure.

b. workload_mix2.sh

The code for the workload_mix2.sh is as follows:

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

We used two workloads in this workload: ***arithoh.sh*** and ***syscall.sh***. We discovered that arithoh.sh and syscall.sh are both CPU constrained after inspecting their sources and scheduling order. arithoh.sh performs CPU-bound arithmetic operations, whereas syscall.sh runs a loop that calls the system.



arithoh.sh and syscall.sh completed

From the figure we can see that syscall.sh, with PID 77 being swapped one after the other depending upon their priority. Here arithoh.sh is completed before syscall.sh

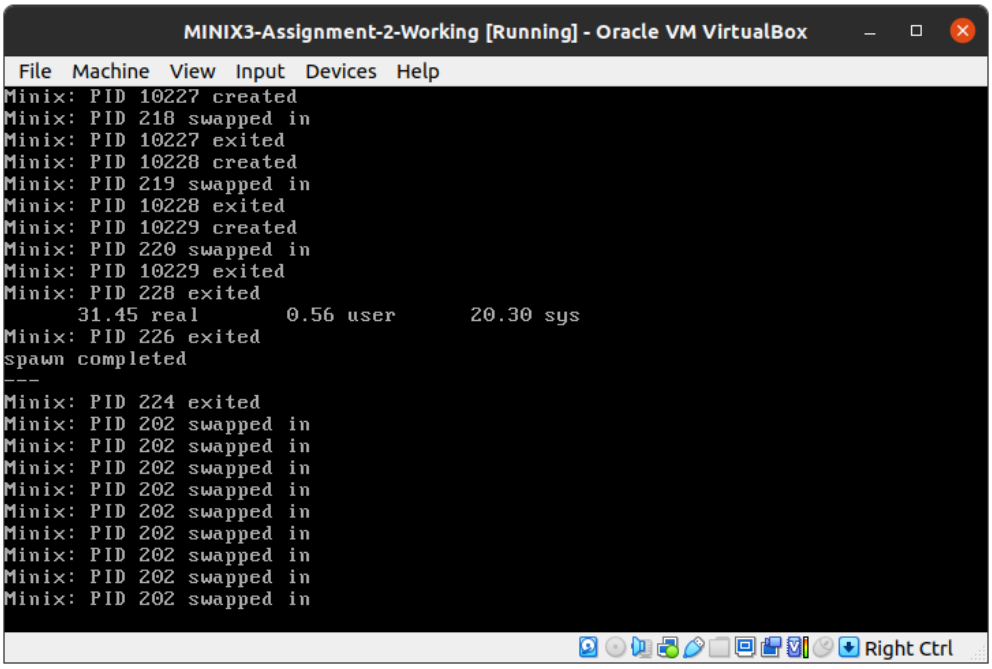
c. workload_mix3.sh

The code for the workload_mix3.sh is as follows:

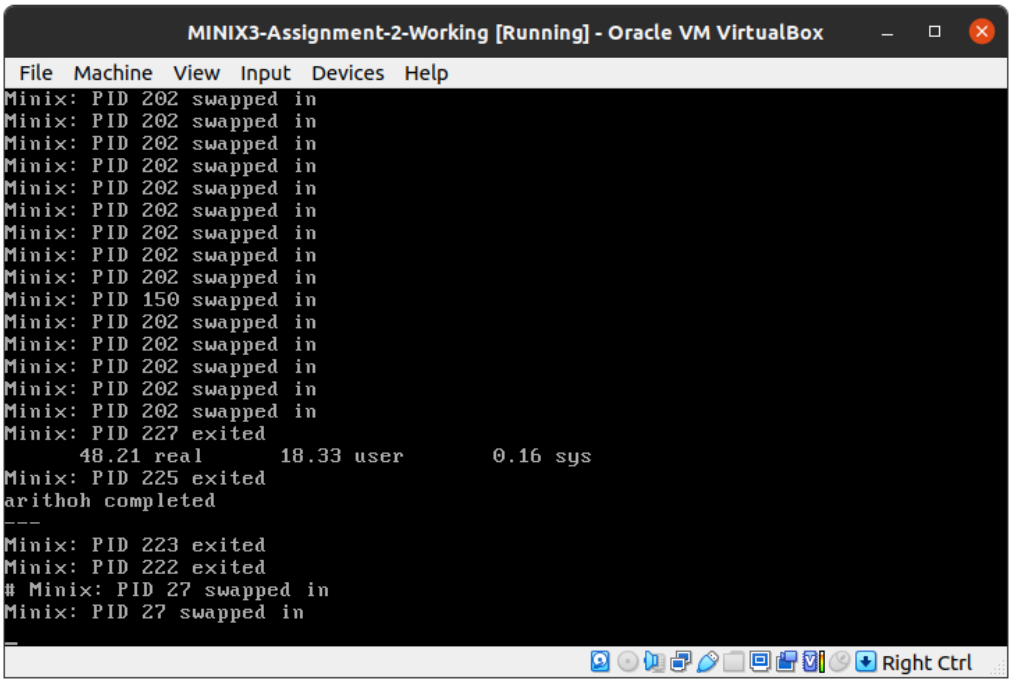
```
#!/bin/sh
./arithoh.sh &
```

```
./spawn.sh &
Wait
```

we utilised two workloads in this workload: ***arithoh.sh*** and ***spawn.sh***. After inspecting their source code and scheduling order, we discovered that arithoh.sh and spawn.sh are both CPU constrained. arithoh.sh performs CPU-bound arithmetic operations, whereas spawn.sh creates (forks) new processes that exit immediately.



spawn.sh completed



arithoh.sh completed

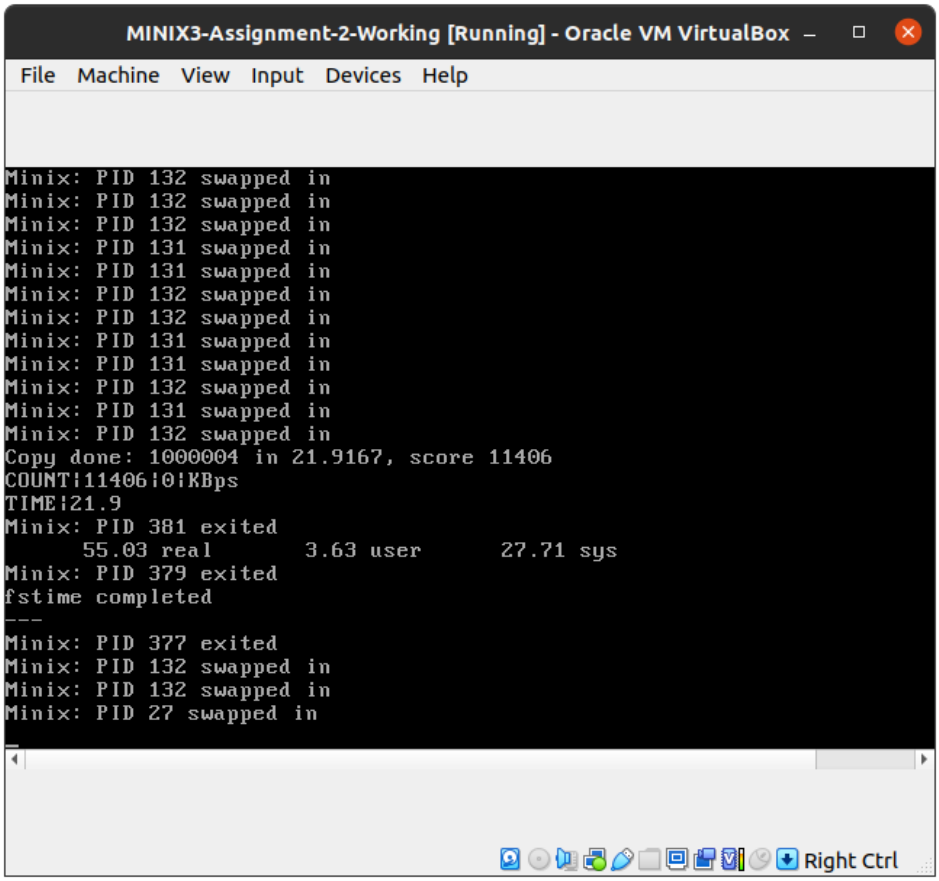
We can see from the above image that spawn.sh is constantly creating new processes and scheduling them to reduce response time. According to the source code, these newly generated processes quit instantly. After a while, spawn.sh and the newly generated process will exit. The arithoh.sh with PID 202 is then scheduled till completion, as seen in the figure.

d. workload_mix4.sh

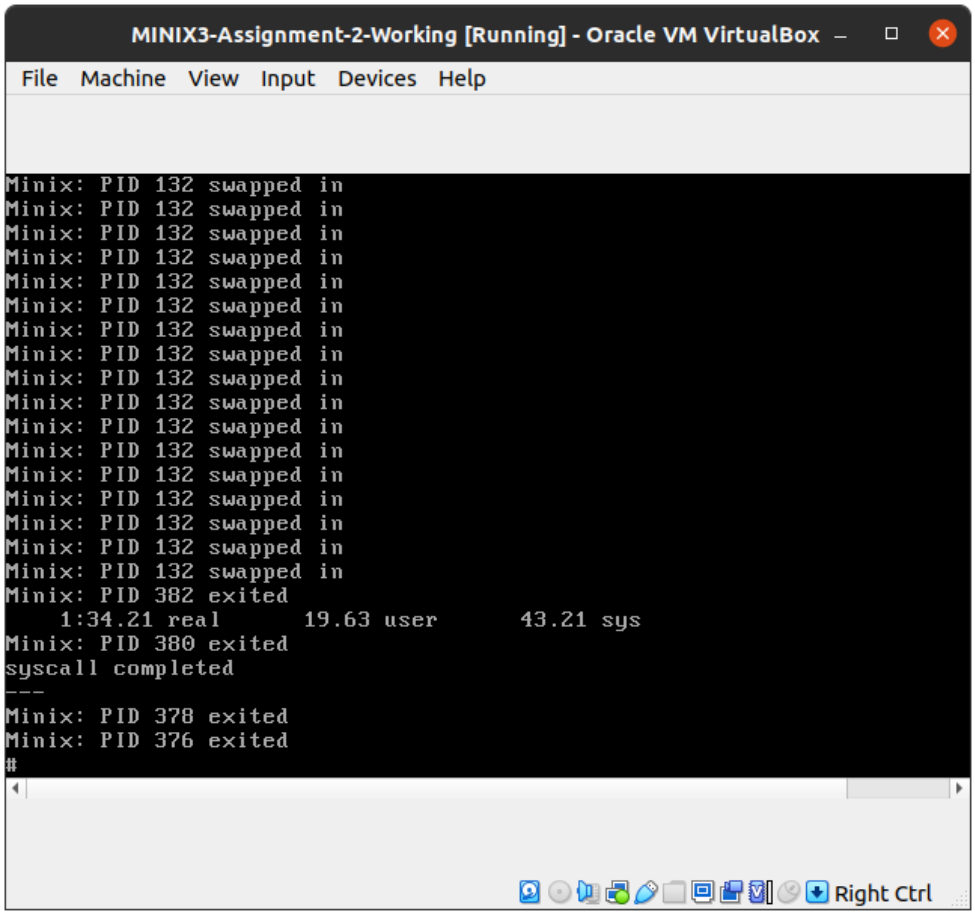
The code for the workload_mix4.sh is as follows:

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

In this workload we have used two workloads, namely ***fstime.sh*** and ***syscall.sh***. After looking at their source and the order of scheduling, we came to know that fstime.sh is I/O bound and syscall.sh is CPU bound.



fstime.sh completed



syscall.sh completed

From the above figure,we can see that fstime.sh, with PID 131 is waiting for I/O response, while syscall.sh, with PID 132 is swapped in. So once fstime.sh is swapped in, it runs until completion. After this, syscall.sh is swapped in and runs until completion.

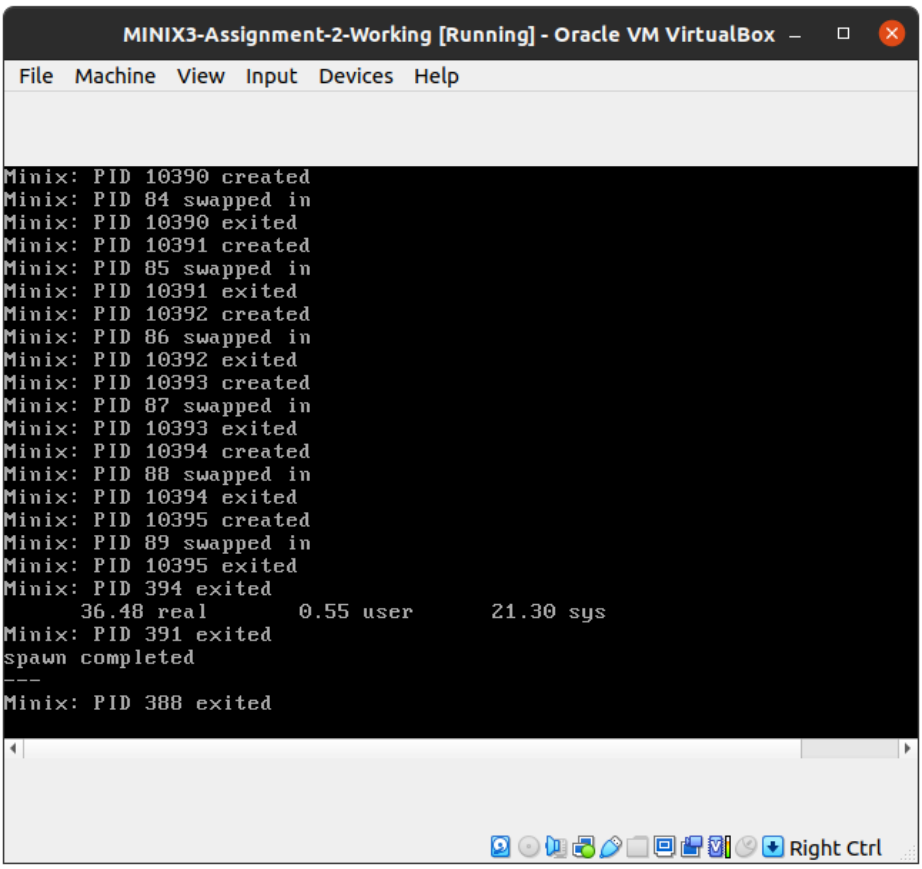
e. workload_mix5.sh

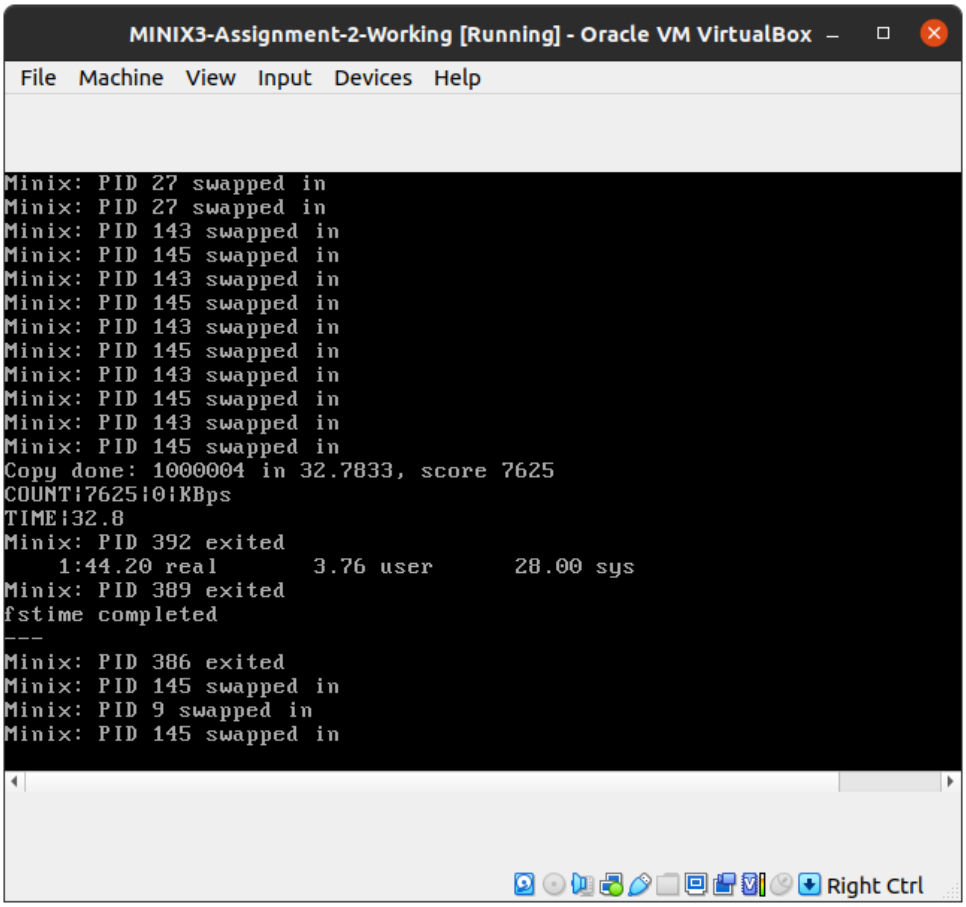
The code for the workload_mix5.sh is as follows:

```
#!/bin/sh
./fstime.sh &
./pipe.sh &
./spawn.sh &
Wait
```

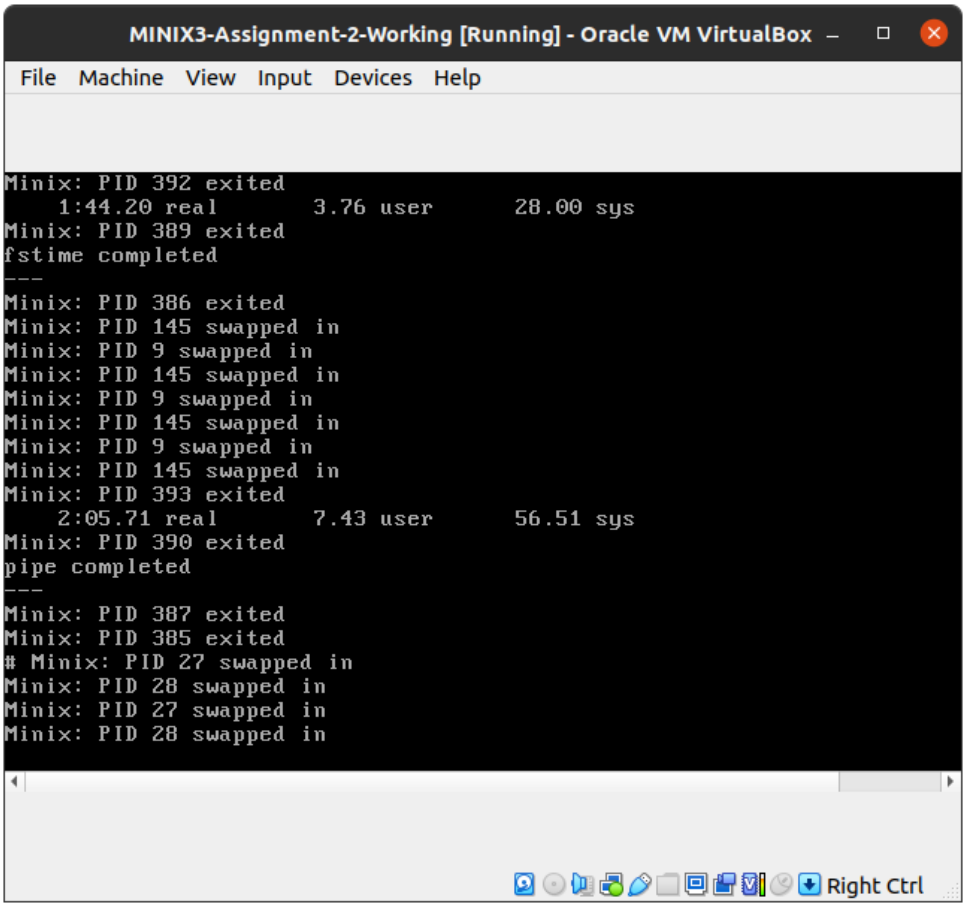
In this workload we have used a three workloads, namely fstime.sh, pipe.sh and spawn.sh. After looking at their source and the order of

scheduling, I came to know that both fstime.sh is I/O bound. The pipe.sh does some I/O bound, spawn.sh is CPU bound.




MINIX3-Assignment-2-Working [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Minix: PID 27 swapped in
Minix: PID 27 swapped in
Minix: PID 143 swapped in
Minix: PID 145 swapped in
Minix: PID 143 swapped in
Minix: PID 145 swapped in
Minix: PID 143 swapped in
Minix: PID 145 swapped in
Minix: PID 143 swapped in
Minix: PID 145 swapped in
Minix: PID 143 swapped in
Minix: PID 145 swapped in
Copy done: 1000004 in 32.7833, score 7625
COUNT¦7625¦0¦KBps
TIME¦32.8
Minix: PID 392 exited
1:44.20 real 3.76 user 28.00 sys
Minix: PID 389 exited
fstime completed

Minix: PID 386 exited
Minix: PID 145 swapped in
Minix: PID 9 swapped in
Minix: PID 145 swapped in
Right Ctrl



spawn.sh with PID 89 is swapped with fstime.sh with PID 143. The fstime.sh is swapped with pipe.sh having PID 145.