CS314 : Operating Systems Lab

Lab 3

Sourabh Bhosale (200010004)
Dibyashu Kashyap (200010013)

January 22, 2023

1 Part 1

Problem

Modify the Minix3 source code such that the string “PID <pid> swapped in” is printed, whenever a user-level process is brought in by the scheduler.

Answer

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 (200010004): 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 runme.sh bash file. Which when run, copies the modified schedule.c file into appropriate location and rebuilds the minix OS.

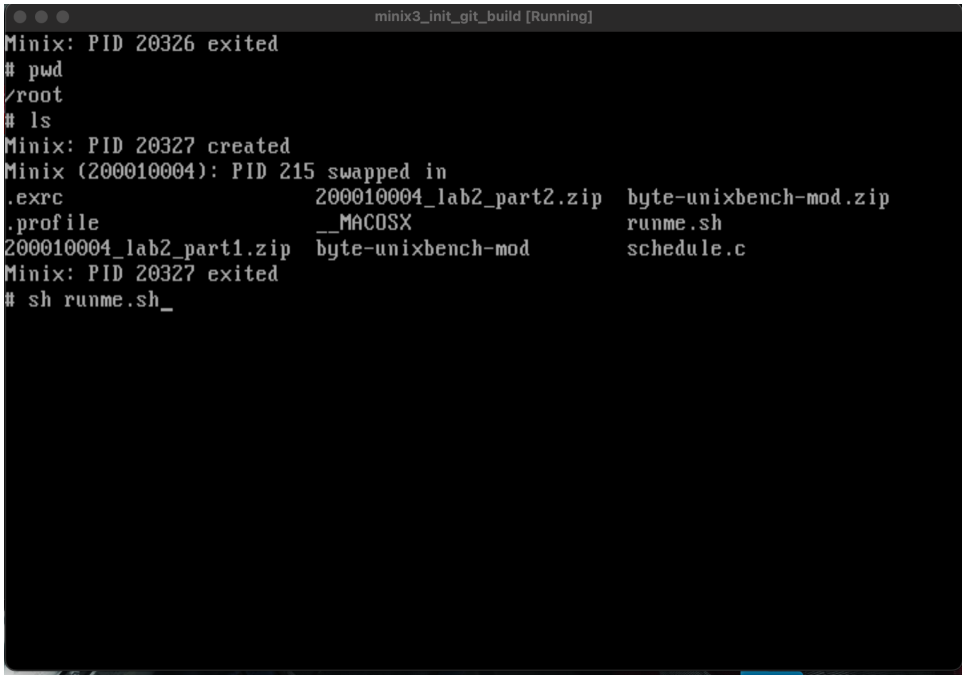

Figure 1:

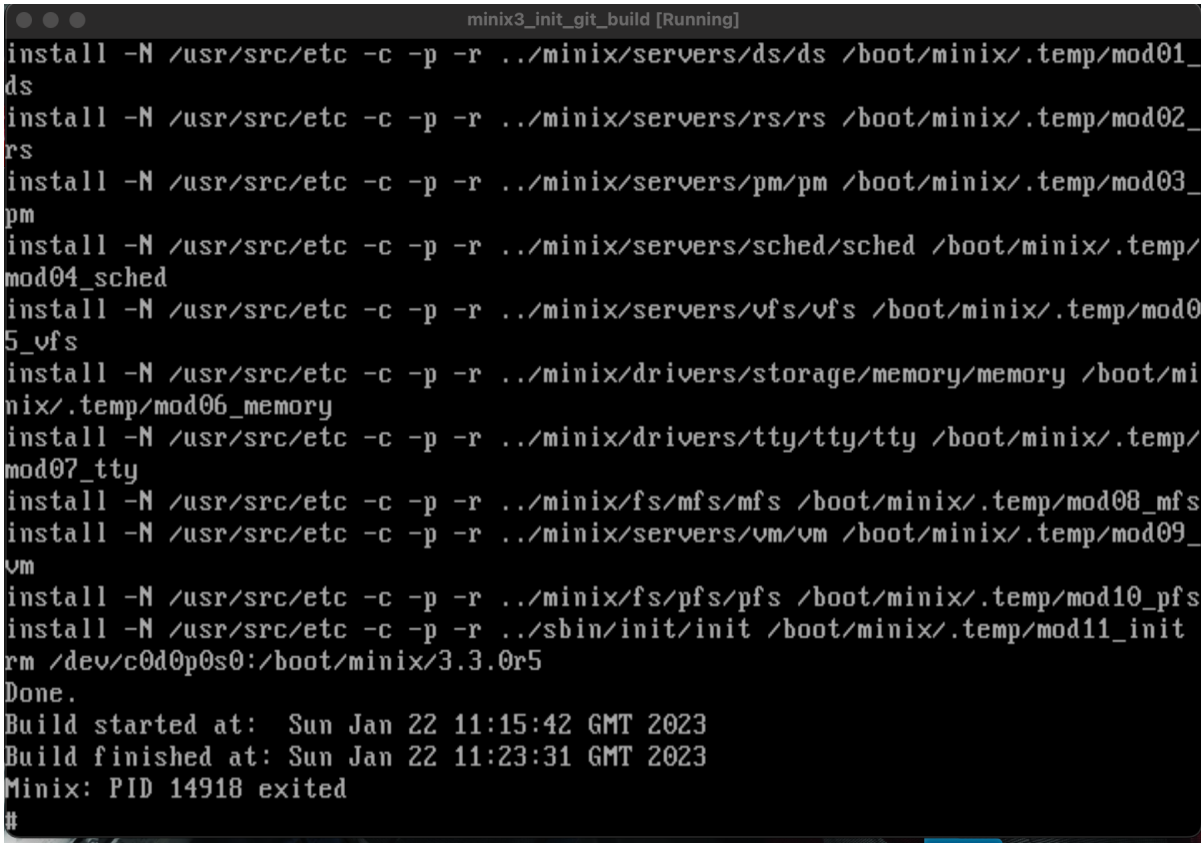
minix3_init_git_build [Running]
install -N /usr/src/etc -c -p -r ../minix/servers/ds/ds /boot/minix/.temp/mod01_
ds
install -N /usr/src/etc -c -p -r ../minix/servers/rs/rs /boot/minix/.temp/mod02_
rs
install -N /usr/src/etc -c -p -r ../minix/servers/pm/pm /boot/minix/.temp/mod03_
pm
install -N /usr/src/etc -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/3.3.0r5
Done.
Build started at: Sun Jan 22 11:15:42 GMT 2023
Build finished at: Sun Jan 22 11:23:31 GMT 2023
Minix: PID 14918 exited
#

Figure 2:

2 Part 2

Problem

You may study the behavior of the scheduler by seeing the sequence of “PID” prints when these workloads are run.

workload_mix1.sh

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

We can expect the following behavior:

The `arithoh.sh` script, is likely to run benchmark tests that measure the performance of the CPU, specifically the overhead of arithmetic operations. examining the code we can easily see there are computationally intensive operations inside two nested loops.

The `fstime.sh` script is likely to run benchmark tests that measure the performance of the file system, specifically the time taken to perform various file operations.

The third command "wait" is used to wait for all background processes to complete. So that the workload mix sh process exits only after all the inside processes are exited.

As both arithoh.sh and fstime.sh are running in parallel and as both are intensive in their own nature, the scheduler will likely have to switch frequently between the two processes in order to allocate resources accordingly. The "wait" command will make sure that the script will wait for both the scripts to complete before exiting.

In terms of schedule orders, the scheduler will likely prioritize the CPU-intensive arithoh.sh process over the I/O-intensive fstime.sh process when the system is under heavy load. However, when the system is less loaded, the scheduler may give more resources to the fstime.sh process in order to complete file operations quickly.

In workload mix1.sh, an instance of file arithoh.sh(217) and an instance of file fstime.sh(218) are run as shown in fig 3-4.

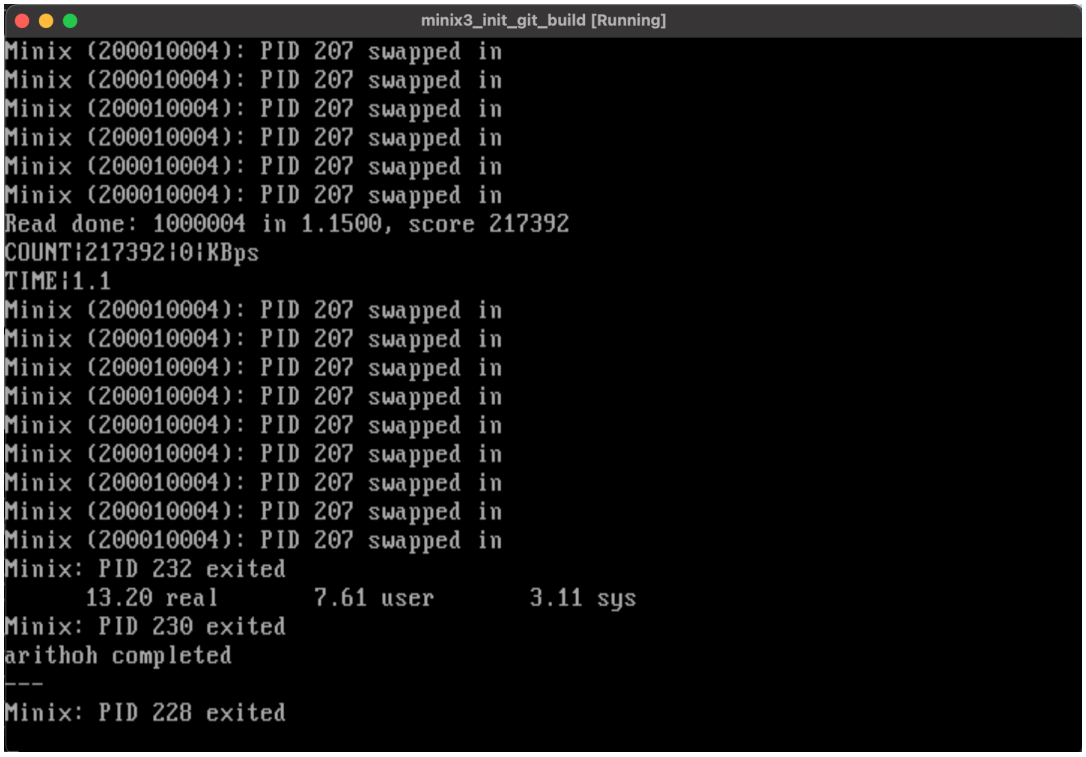

Figure 3:

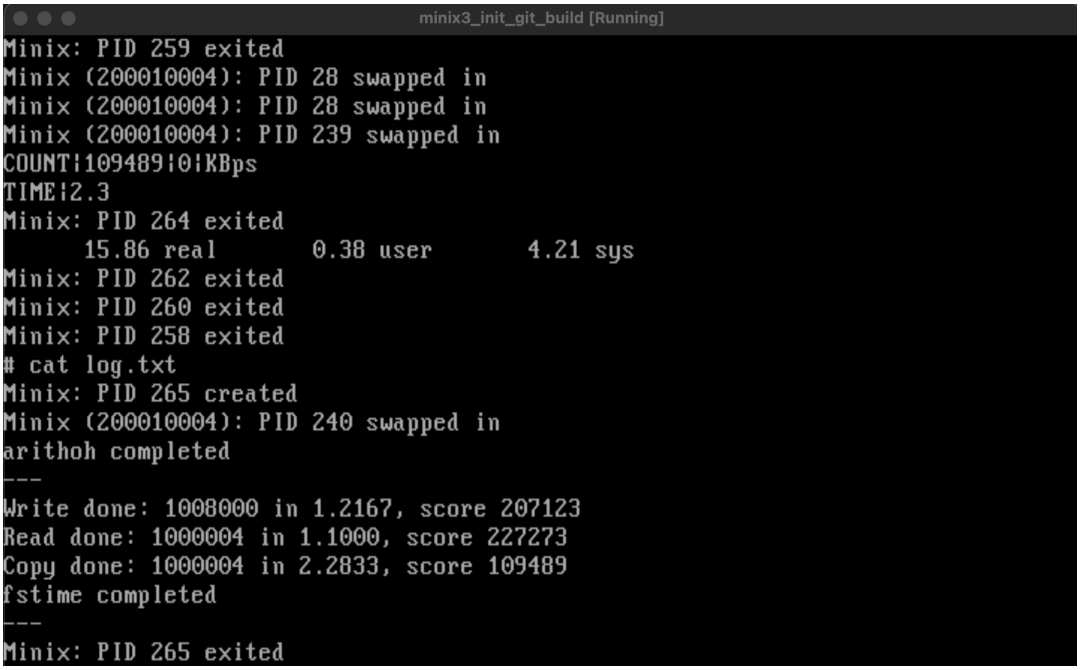

Figure 4:

workload_mix2.sh

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

We can expect the following behavior:

The `./arithoh.sh` script in the background. This script is likely to run benchmark tests that measure the performance of the CPU, specifically the overhead of arithmetic operations.

The second command, `./arithoh.sh` runs another instance of the arithoh.sh script in the background.

The third command, `./arithoh.sh` runs another instance of the arithoh.sh script in the background. Now, the scheduler have to switch frequently between three processes.

As all the three instances of arithoh.sh are running in parallel and as all are intensive in their own nature, the scheduler will have to switch frequently between the three processes in order to allocate resources accordingly.

In here, first arithoh.sh has PID of 248, second arithoh.sh has PID 249 and the last arithoh.sh has PID 250. Clearly, if multiple instances are run of arithoh.sh which are CPU Intensive in nature as in this workload, it can be seen that th's getting scheduled alternatively. Process with PID 248, 249, 250 correspond to 3 instances of workload in arithoh.sh that are scheduled alternatively until they are completed as shown in fig 5 and 6.

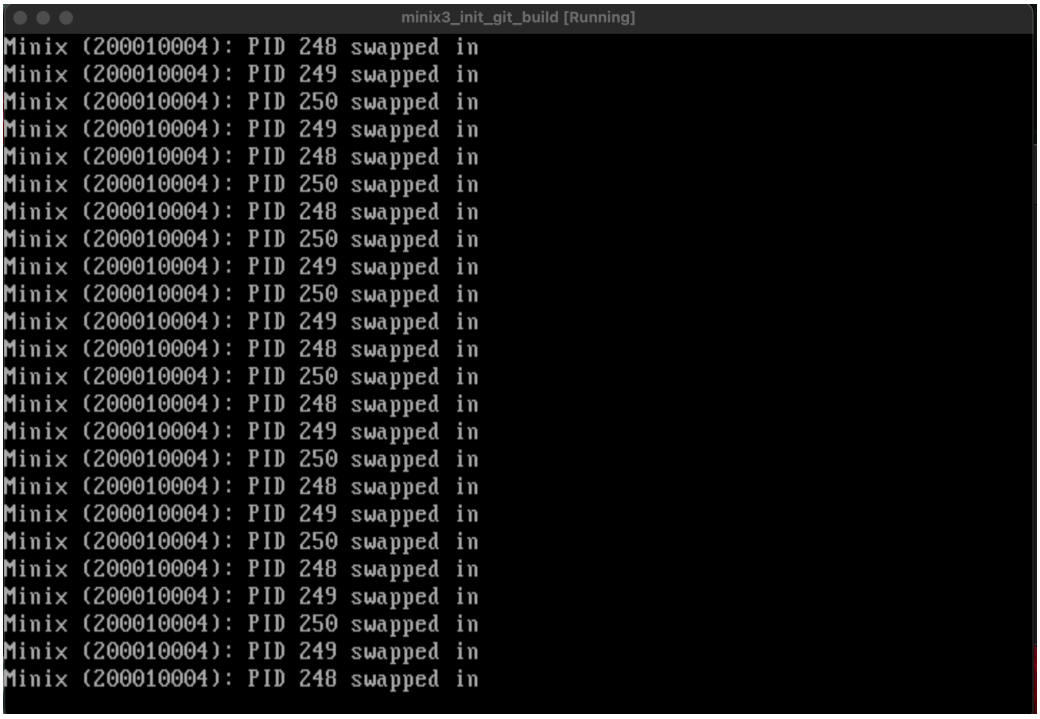

Figure 5:

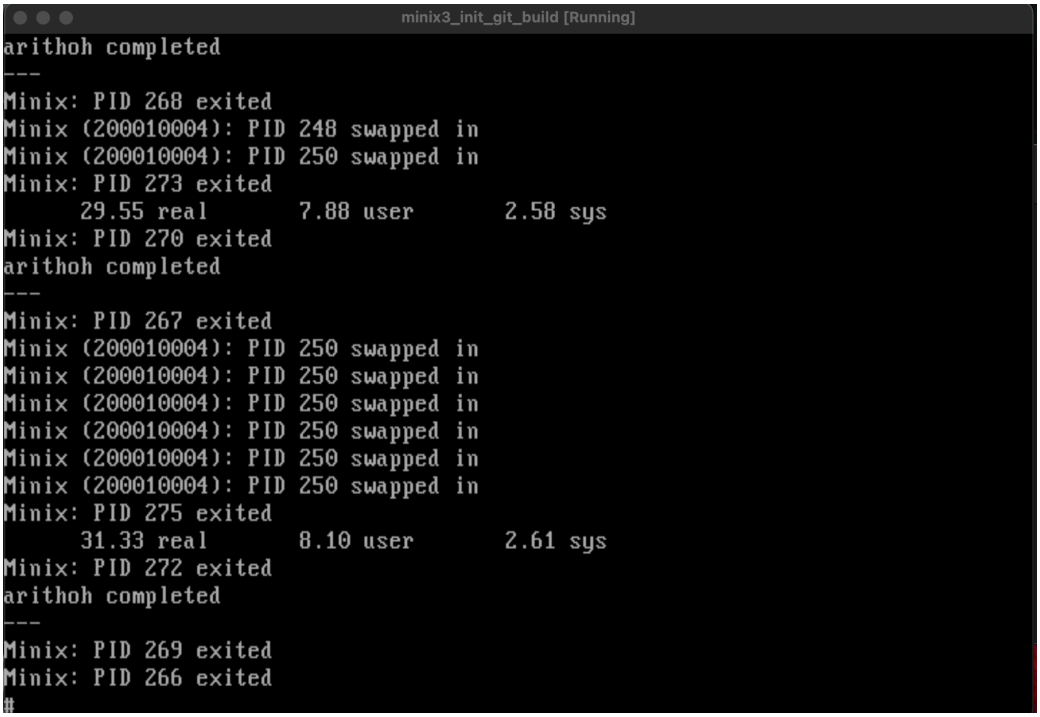

Figure 6:

workload_mix3.sh

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

We can expect the following behavior:

The first command, `./arithoh.sh` , runs the arithoh.sh script in the background.

The second command, `./syscall.sh` runs the syscall.sh script in the background. This script is likely to run benchmark tests that measure the performance of system calls, specifically the overhead of system calls.

As both arithoh.sh and syscall.sh are running in parallel and as both are intensive in their own nature, the scheduler will likely have to switch frequently between the two processes in order to allocate resources accordingly. The "wait" command will make sure that the script will wait for both the scripts to complete before exiting.

In terms of schedule orders, the scheduler will likely prioritize the CPU-intensive arithoh.sh process over the syscall-intensive syscall.sh process when the system is under heavy load. However, when the system is less loaded, the scheduler may give more resources to syscall.sh process in order to complete system calls quickly. as shown in fig 8-11.

In here, arithoh.sh has PID 7 and syscall.sh has PID 11 as shown in fig 7-9.

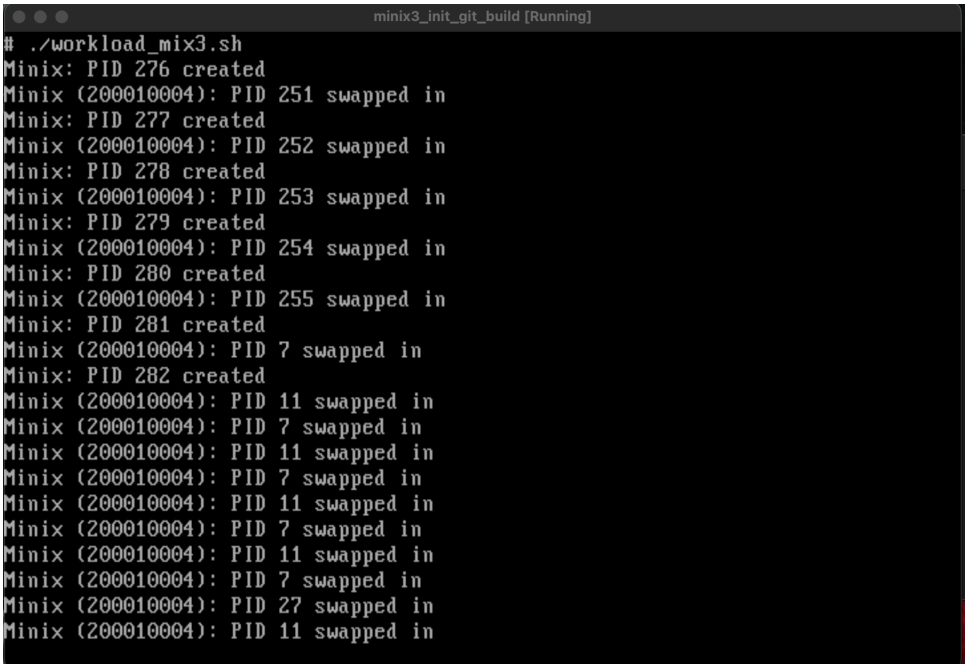

Figure 7:

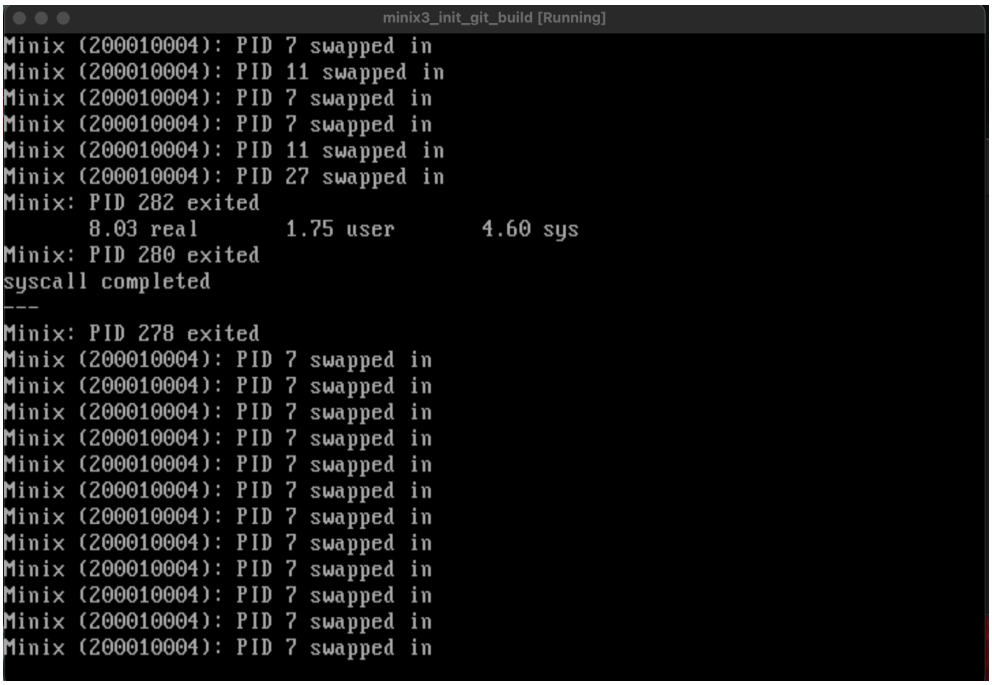

Figure 8:

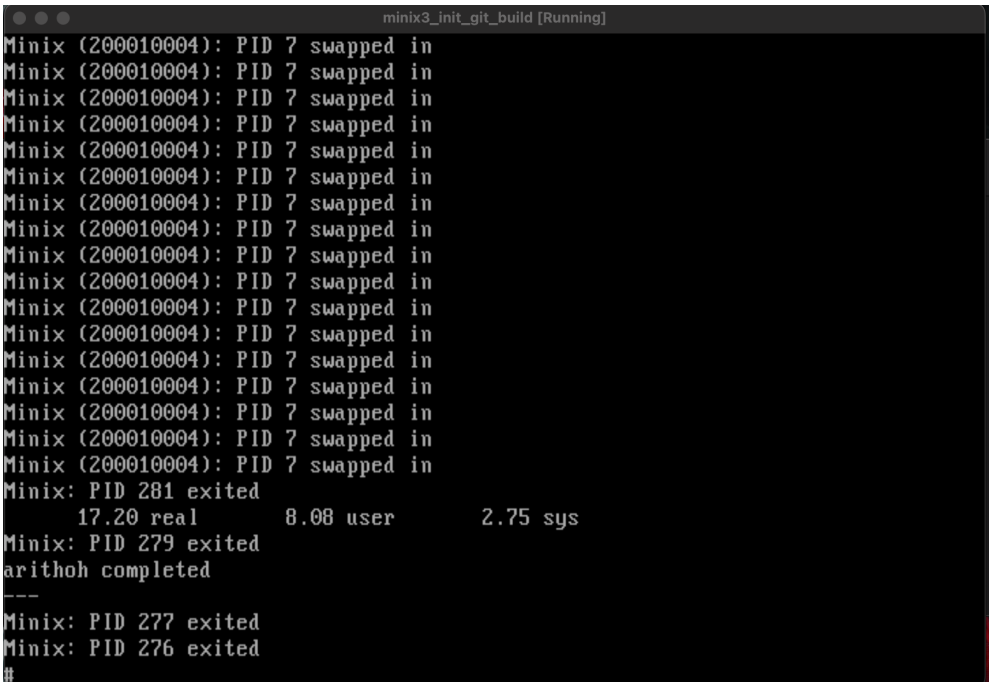

Figure 9:

workload_mix4.sh

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

We can expect the following behavior:

The first command, `./fstime.sh` runs the fstime.sh script in the background. This script is likely to run benchmark tests that measure the performance of the file system, specifically the time taken to perform various file operations.

The second command, `./fstime.sh` , runs another instance of the fstime.sh script in the background. As this script is also I/O-intensive, the scheduler will have to switch frequently between the two processes in order to allocate resources accordingly.

The third command, `./fstime.sh` , runs another instance of the fstime.sh script in the background. Now, the scheduler have to switch frequently between three processes.

The fourth command "wait" is used to wait for all background processes to complete.

As all the three instances of fstime.sh are running in parallel and as all are I/O-intensive in their own nature, the scheduler will have to switch frequently between the three processes in order to allocate resources accordingly. The "wait" command will make sure that the script will wait for all the three scripts to complete before exiting.

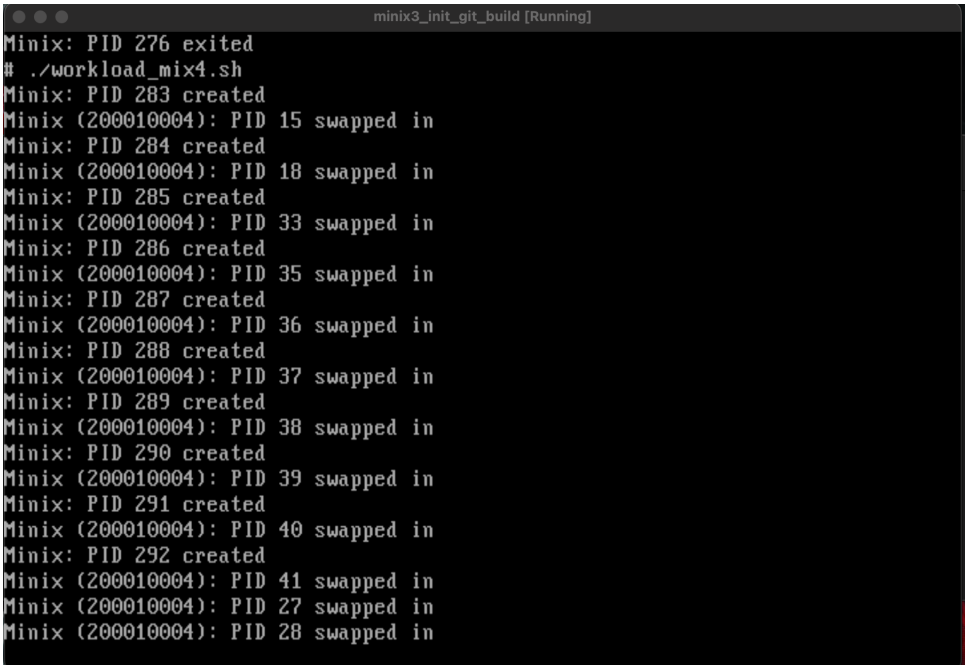

Figure 10:

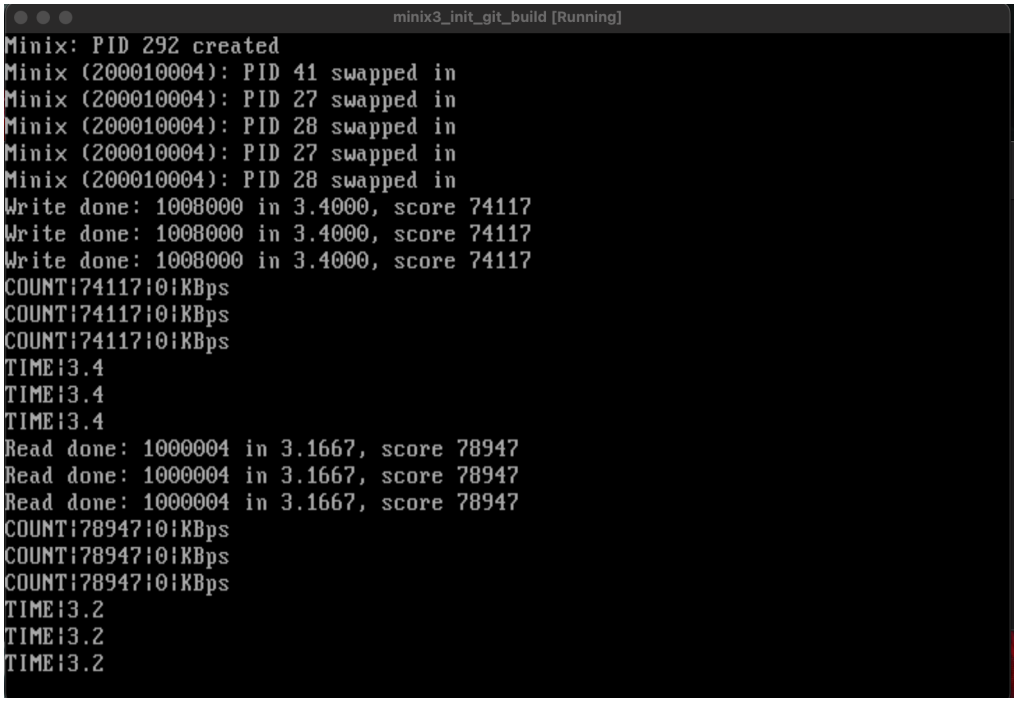

Figure 11:

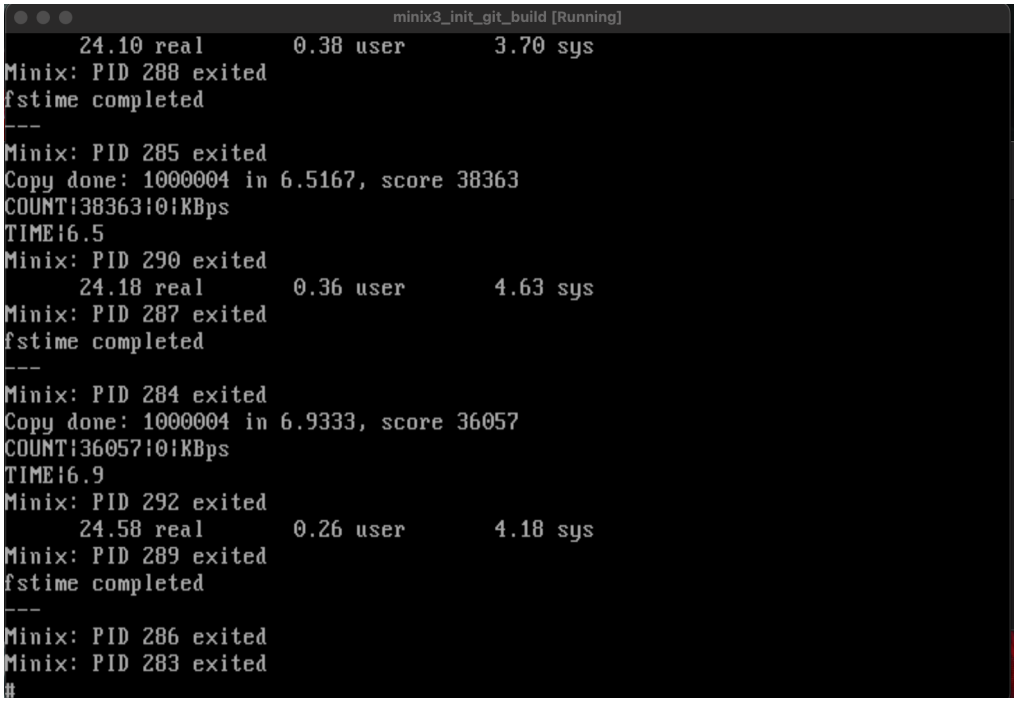

Figure 12:

workload_mix5.sh

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

We can expect the following behavior:

The first command, `./arithoh.sh` , runs the arithoh.sh script in the background.

The `pipe.sh` script is likely to run benchmark tests that measure the performance of the file system, specifically the time taken to perform various file operations.

As both arithoh.sh and pipe.sh are running in parallel and as both are intensive in their own nature, the scheduler will likely have to switch frequently between the two processes in order to allocate resources accordingly.

In terms of schedule orders, the scheduler will likely prioritize the CPU-intensive arithoh.sh process over the pipe-intensive pipe.sh process when the system is under heavy load. However, when the system is less loaded, the scheduler may give more resources to pipe.sh process in order to complete pipe operations quickly. 93 is the PID of arithoh.sh and pipe.sh has PID 94 as shown in fig 13-14

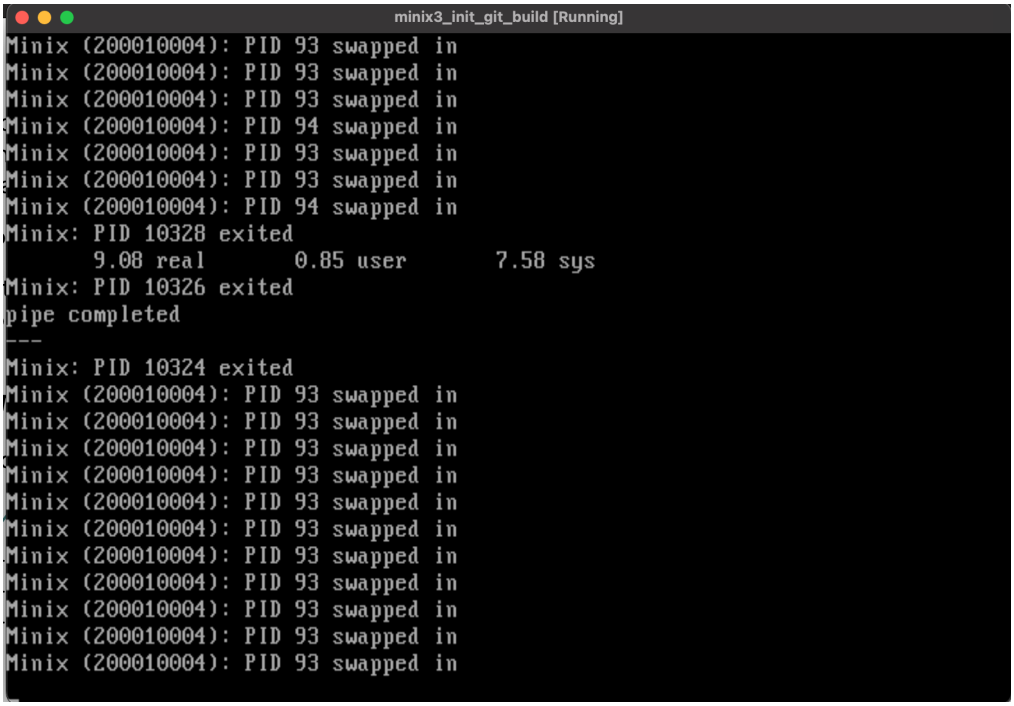

Figure 13:

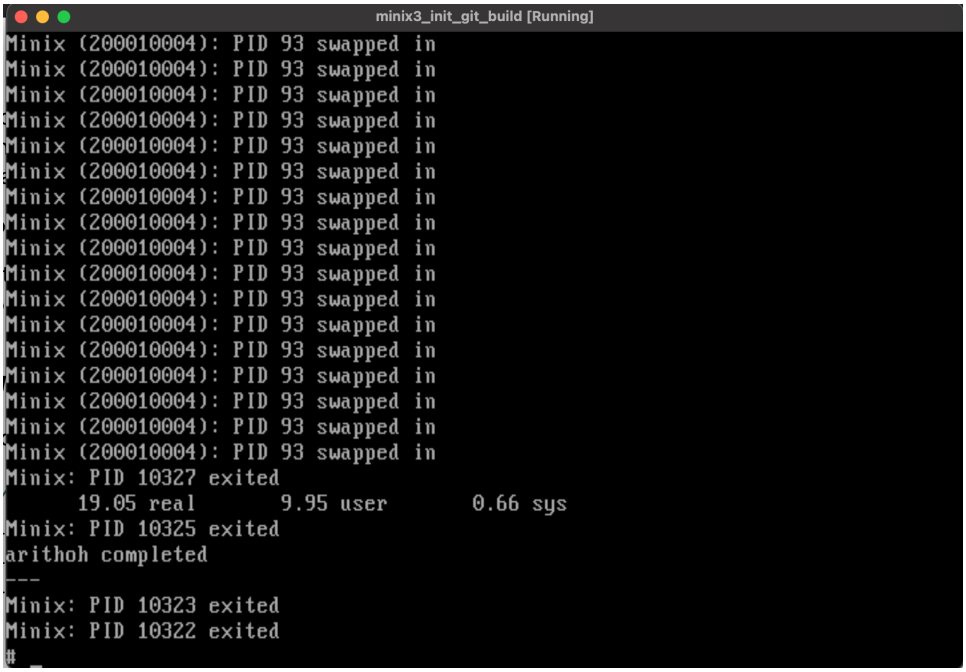

Figure 14:

workload_mix6.sh

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

In this workload, we have used two workloads, namely `arithoh.sh` and `spawn.sh` . After looking at their source code and the order of scheduling,We 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.

From the figure, 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 with along with the newly created process terminate as seen from figure 16-18.

Then, the arithoh.sh, with PID 65 is scheduled until completion as seen from figure 15-17.

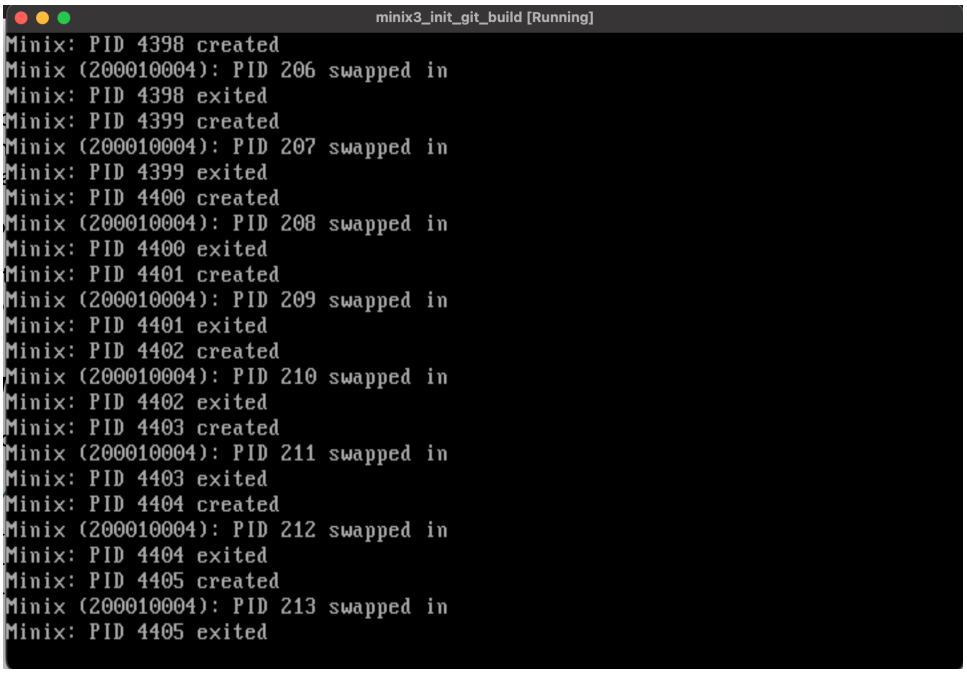

Figure 15:

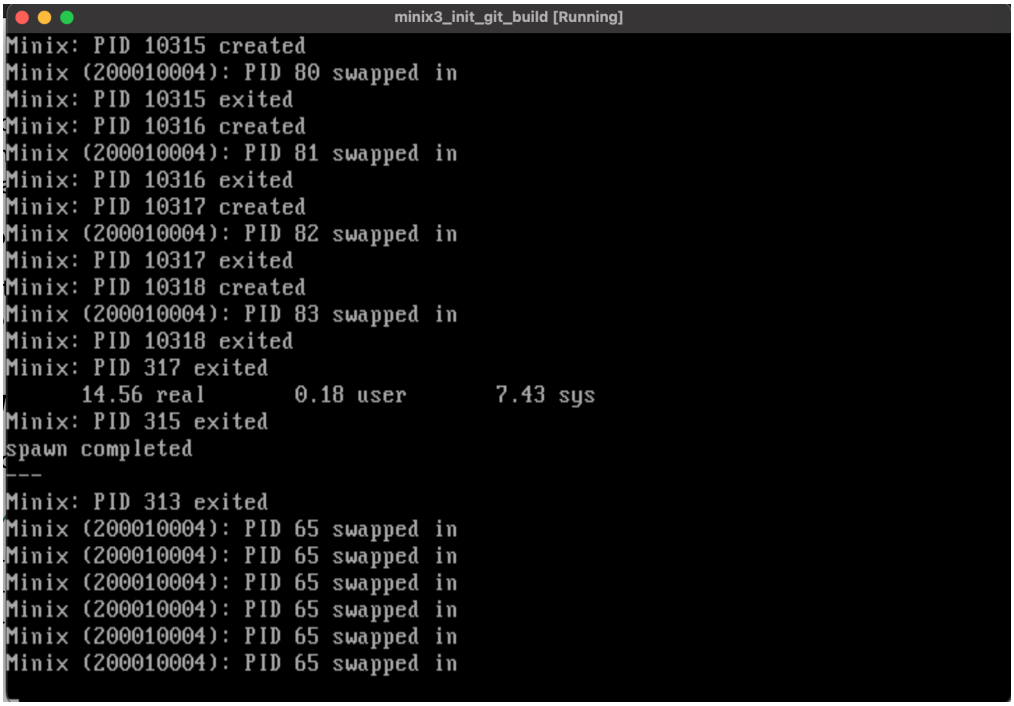

Figure 16:

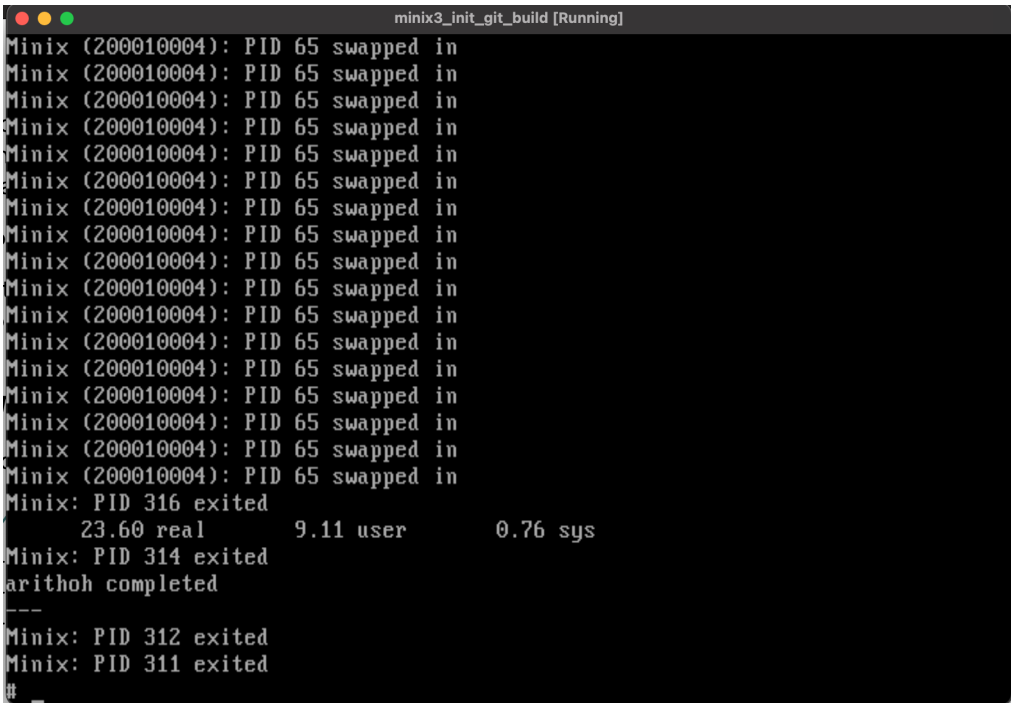

Figure 17: