CS 314 – Operating Systems Lab
Lab–3 Report

Student 1: Sahaja Nandyala
Roll-No: 200010032

Student 2: Kavali Sri Vyshnavi Devi
Roll-No: 200010023

1 Part I

The schedule.c file in the location usr/src/minix/servers/sched has been made following changes for printing "PID pid swapped in", whenever a user-level process is brought in by the scheduler.

The following if statement has been added to "schedule_process" function in schedule.c file. Here the USER_Q signifies ((MIN_USER_Q - MAX_USER_Q) / 2 + MAX_USER_Q) which is specified in **config.h** file. Datatype of _ENDPOINT_P is defined in **proc.c** file.

```
//------------------------------------------------------------------------------
if (rmp->priority >= USER_Q){
        printf("Minix(200010032): PID %d swapped in\n", _ENDPOINT_P(rmp->endpoint));
}
//------------------------------------------------------------------------------
```

Figure 1: Part1 code

To build the changes on MINIX3 OS, a run.sh is also made which copies schedule.cs from the present working directory to usr/src/minix/servers/sched and building the OS again.

```
echo "copying files"
cp schedule.c /usr/src/minix/servers/sched/schedule.c
echo "going to src directory and building the updated code"
cd /usr/src
make build MKUPDATE=yes >log.txt 2>log.txt
echo "build completed successfully"
exit 0
```

The screenshot of build success for Part1.

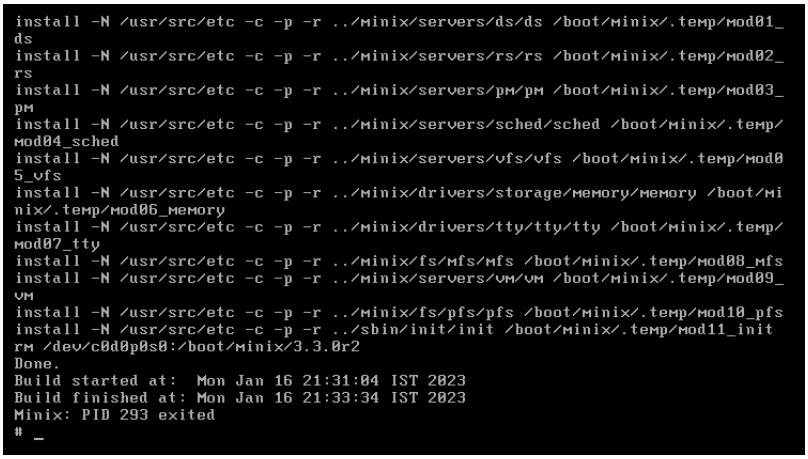

Figure 2: Build Successful

Later reboot the OS on successful build to observe the changes. Executing printenv command after rebuilding the OS.

Figure 3: Executing printenv command

Executing ls command after rebuilding the OS.

Figure 4: Executing ls command

2 Part II

The source code for UnixBench benchmark suite was copied into the home folder in the guest Minix3 VM and built using gmake command. Later six work mix files are written which tries various combinations of UnixBench programs to study.

```
Minix: PID 290 exited
# pwd
/home/byte-unixbench-mod/byte-unixbench-mod/UnixBench
# gmake
```

Figure 5: gmake command

```
Minix: PID 320 created
MINIX(200010023): PID 69 swapped in
Minix: PID 320 exited
Minix: PID 321 created
MINIX(200010023): PID 70 swapped in
Minix: PID 321 exited
Minix: PID 319 exited
clang -o pgms/whetstone-double -Wall -pedantic -O0 -ffast-math -I ./src -DTIME -
DDP -DGTODay -DUNIXBENCH src/whets.c -lm
Minix: PID 322 created
MINIX(200010023): PID 71 swapped in
Minix: PID 323 created
MINIX(200010023): PID 72 swapped in
Minix: PID 323 exited
Minix: PID 324 created
MINIX(200010023): PID 73 swapped in
Minix: PID 324 exited
Minix: PID 322 exited
gmake[1]: Leaving directory '/home/byte-unixbench-mod/UnixBench'
Minix: PID 266 exited
Minix: PID 254 exited
```

Figure 6: Executing gmake command

2.1 Study On Programs In Src Folder

- **arith.c(aritoh.sh)** long arithmetic operations more CPU bound operations.
- **fstime.c(fstime.sh)** IO performance is measured using this. file copy, 1024 byte buffer size, 500 max blocks.
- **pipe.c(pipe.sh)** The pipe benchmark measures the number of times a process can write 512 bytes to a pipe and read them back per second. The pipe switching benchmark measures the number of times two processes can exchange an increasing integer through a pipe. Here more context switches occur.
- **spawn.c(spawn.sh)** It does fork and write to files both CPU and I/O bound.
- **syscall.c(syscall.sh)** Sit in a loop calling getpid where constant switching between user space and kernel also cause some overhead.

2.2 Study of Programs by workload mix

Using the UnixBench Benchmark Suite, some workload mix shell scripts were created in order to study the behavior of the scheduler by observing the sequence in which processes are swapped in.

2.2.1 workload_mix1.sh

In this shell script we have used **aritoh.sh** and **fstime.sh** as following code

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

Figure 7: workload_mix1.sh code

Here, PID of aritoh is 201 and PID of fstime is 202. We have noticed that the aritoh.sh is doing big arithmetic operations which require CPU time long, where as fstime.sh mostly perform I/O operations for which the requirement of CPU is less but goes to blocked state frequently. So, when fstime.sh is waiting for response from I/O device aritoh.sh is being executed which that is why we can observe that it is getting swapped more frequently.

Also airtoh.sh took more time to execute and so it is exited later as compared to fstime.sh as their dependency on CPU differs.

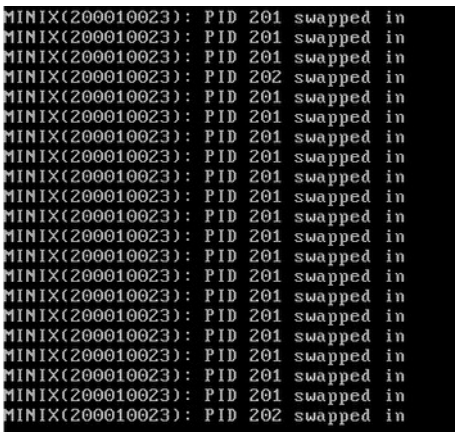

Figure 8: workmix_load1: processes getting swapped

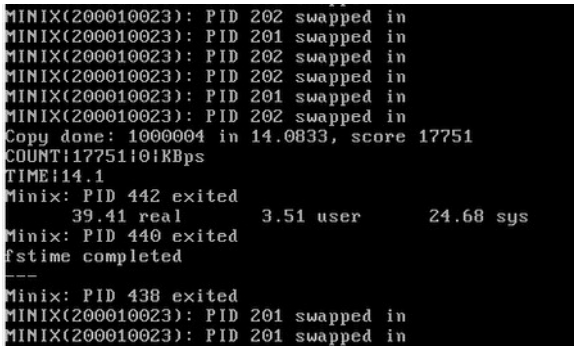

Figure 9: workmix_load1: exit of fstime process

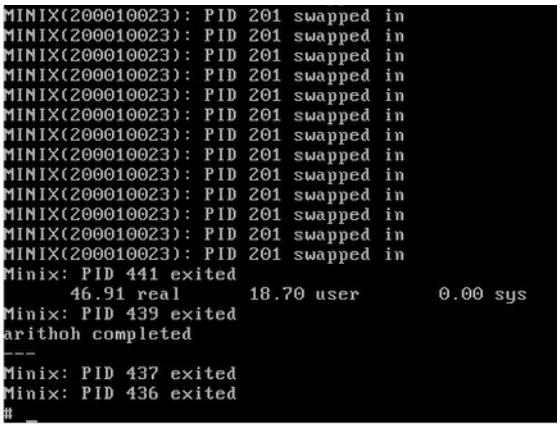

Figure 10: workmix_load1: exit of aritoh process

2.2.2 workload_mix2

In this shell script we have used **aritoh.sh** and **syscall.sh** as following

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

Figure 11: workload_mix2.sh code

Here, PID of aritoh is 213 and PID of syscall is 214. we can observe that aritoh.sh is meant for making big arithematic operations where as syscall.sh calls kernel functions that is getpid() which makes more context switches as compared to aritoh.sh. Here aritoh.sh and syscall.sh are being swapped in alternatively depending on their intensity of consuming the CPU. At last syscall.sh was completed as it is much CPU intense work as compared to aritoh.sh which completed execution earlier. So, from this we can interpret that the cost taken to perform context switches in syscall.sh is more than the CPU time taken by aritoh.sh which we can tell by the order of completion of 2 processes.

```
MINIX(200010023): PID 213 swapped in
MINIX(200010023): PID 213 swapped in
MINIX(200010023): PID 214 swapped in
MINIX(200010023): PID 213 swapped in
MINIX(200010023): PID 214 swapped in
MINIX(200010023): PID 213 swapped in
MINIX(200010023): PID 213 swapped in
MINIX(200010023): PID 213 swapped in
MINIX(200010023): PID 213 swapped in
MINIX(200010023): PID 213 swapped in
Minix: PID 453 exited
     59.88 real      18.80 user       0.00 sys
Minix: PID 451 exited
arithoh completed
---
Minix: PID 449 exited
```

Figure 12: workload_mix2: exit of aritoh process

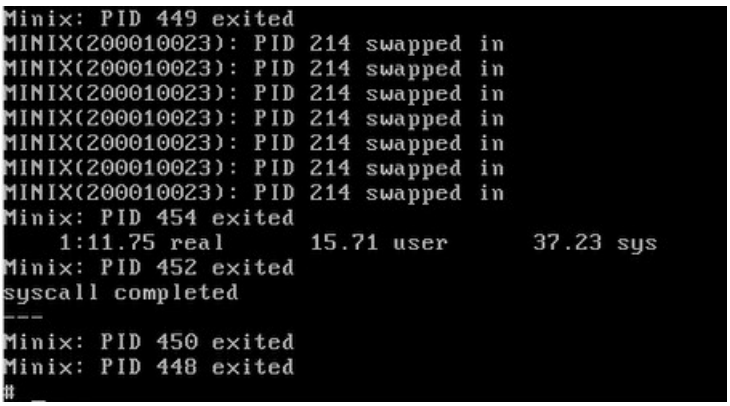

Figure 13: workload_mix2: exit of syscall process

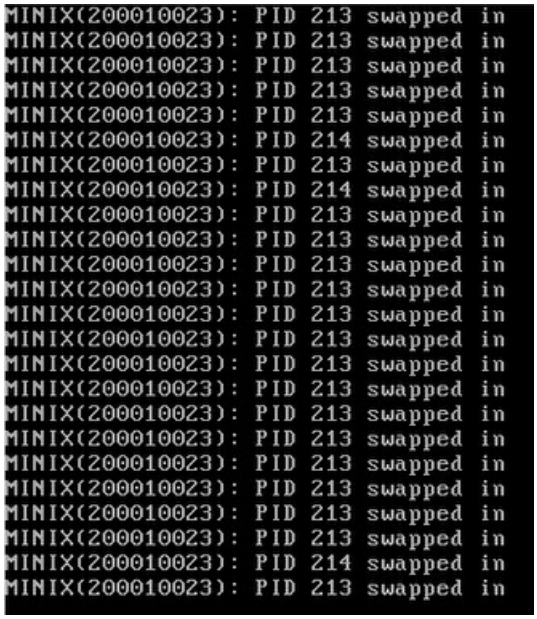

Figure 14: work2-swap

2.2.3 workload_mix3

In this shell script we have used **pipe.sh** and **fstime.sh** as following //

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

Figure 15: workload_mix3.sh code

Here, PID of pipe is 55 and PID of fstime is 57 .We can observe that pipe.sh makes more system calls and make more context switches, where as fstime.sh is an I/O bound. So, when fstime.sh is present in the blocked state that is waitting for an interrupt from I/O device pipe.sh can be scheduled that is why pipeline (55) is being swapped in more frequently.
Also as the fstime.sh is I/O bound it is completed first and later pipe.sh completed execution.

```
MINIX(200010023): PID 55 swapped in
MINIX(200010023): PID 55 swapped in
MINIX(200010023): PID 57 swapped in
MINIX(200010023): PID 57 swapped in
MINIX(200010023): PID 55 swapped in
MINIX(200010023): PID 55 swapped in
MINIX(200010023): PID 57 swapped in
MINIX(200010023): PID 57 swapped in
MINIX(200010023): PID 55 swapped in
MINIX(200010023): PID 57 swapped in
MINIX(200010023): PID 55 swapped in
MINIX(200010023): PID 57 swapped in
Copy done: 1000004 in 27.8000, score 8992
COUNT¦8992¦0¦KBps
TIME¦27.8
Minix: PID 523 exited
    1:05.90 real       3.06 user      25.46 sys
Minix: PID 522 exited
fstime completed
---
Minix: PID 518 exited
```

Figure 16: work3 fstime exit

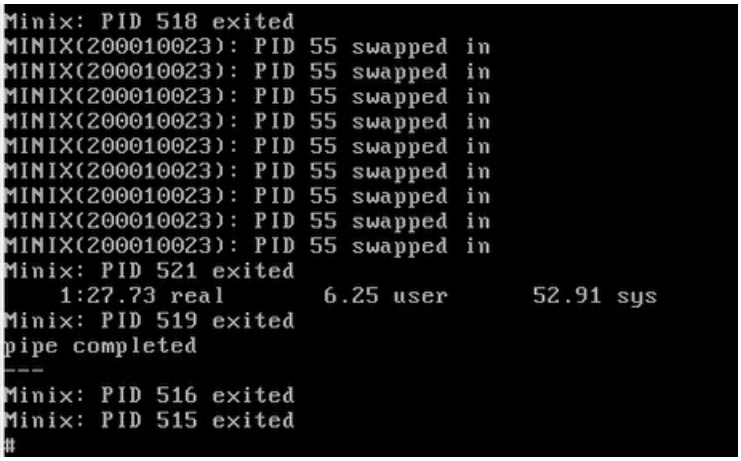

Figure 17: work3 pipe exit

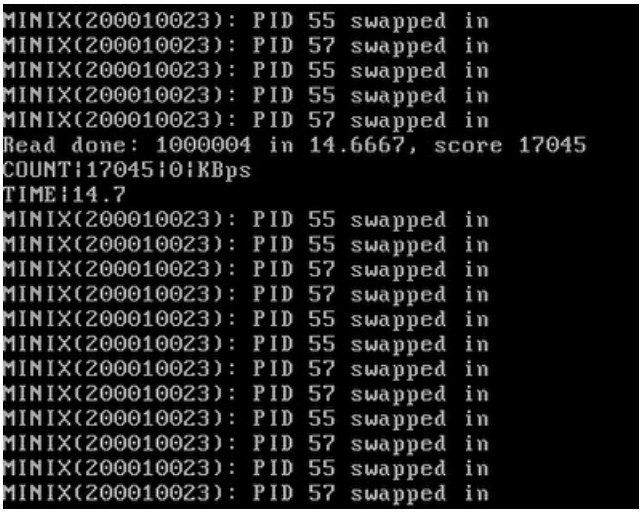

Figure 18: work3 swap

2.2.4 workload_mix4

In this shell script we have used **spawn.sh** as following

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

Figure 19: workload_mix4.sh code

Here, spawn do many fork system calls due to which many context switches happens between child process and parent process by which many process id's are created and so one can notice that many different pid's are showing while a process is getting swapped in. As many context switches happened in spawning it also require more CPU time in saving and restoring data such as registers, program counters, kernel stacks and virtual memory.

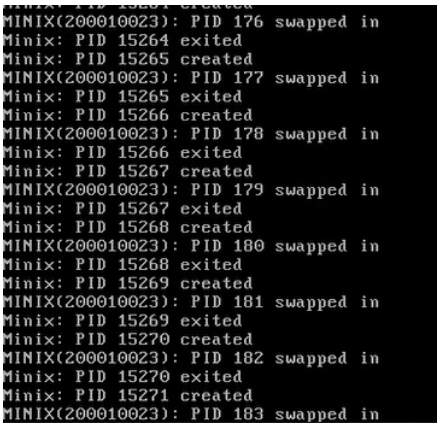

Figure 20: workload_mix4 swapping

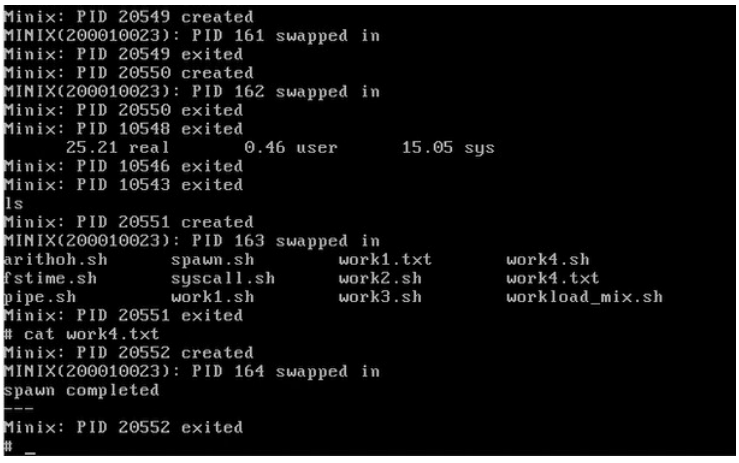

Figure 21: workload_mix4 spwan exit

2.2.5 workload_mix5

In this shell script we have scheduled two **aritoh.sh** jobs are executed as following

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

Figure 22: workload_mix5.sh code

Here, the PID of first aritoh is 35 and the PID of second aritoh is 36. As the jobs that are schedule do the same functionality there shouldn't be change in the intensity on which they use CPU. So, here both the processes are being swapped in alternatively. Also all the processes scheduled here are CPU intense as they perform long long arithmetic operations. All of them are scheduled alternatively till both the processes complete execution.

```
MINIX(200010023): PID 35 swapped in
MINIX(200010023): PID 35 swapped in
MINIX(200010023): PID 35 swapped in
MINIX(200010023): PID 35 swapped in
MINIX(200010023): PID 35 swapped in
MINIX(200010023): PID 35 swapped in
MINIX(200010023): PID 35 swapped in
MINIX(200010023): PID 35 swapped in
Minix: PID 286 exited
     36.98 real      18.60 user       0.00 sys
Minix: PID 284 exited
arithoh completed
---
Minix: PID 282 exited
Minix: PID 281 exited
#
```

Figure 23: work5-1st-aritoh-exit

Figure 24: work5-2nd-aritoh-exit

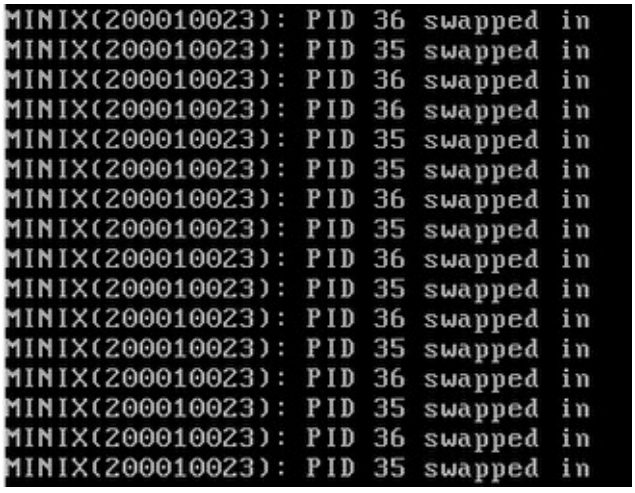

Figure 25: work5-swap

2.2.6 workload_mix6

In this shell script we have schedule three **fstime.sh** jobs are executed as following

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

Figure 26: workload_mix6.sh code

Here the PID of first, second, third fstime are 52, 54, 55 respectively. As the jobs are same there shouldn't be change in the intensity on which they use CPU. So, here all the three processes are being swapped in alternatively. Also all the processes scheduled here are I/O dependent, one can confirm this by observing the execution in which all the processes will be waiting to receive interrupt from I/O device. All of them are scheduled alternatively till all the processes complete execution.

```
MINIX(200010023): PID 24 swapped in
MINIX(200010023): PID 25 swapped in
Copy done: 1000004 in 43.3500, score 5767
COUNT|5767|0|KBps
TIME|43.4
Minix: PID 303 exited
    1:36.36 real       3.28 user      25.91 sys
Minix: PID 301 exited
fstime completed
---
Minix: PID 298 exited
Copy done: 1000004 in 43.6167, score 5731
COUNT|5731|0|KBps
TIME|43.6
Minix: PID 305 exited
    1:36.90 real       3.35 user      23.50 sys
Minix: PID 302 exited
fstime completed
---
Minix: PID 299 exited
```

Figure 27: work6-execution

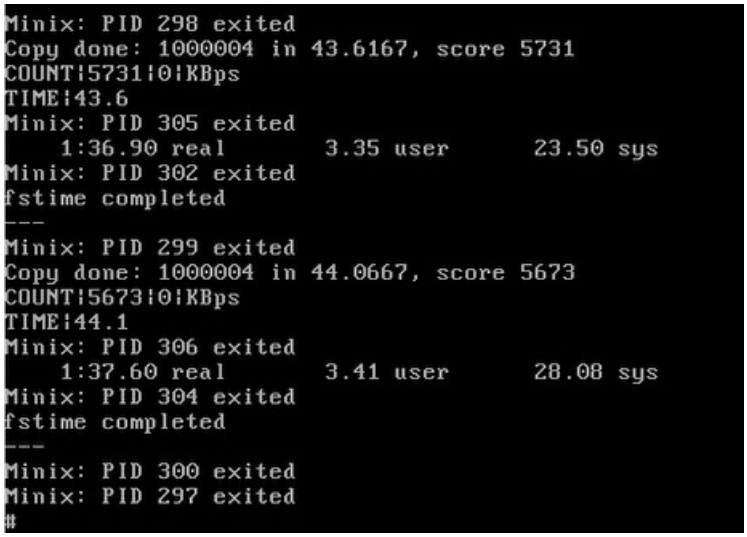

Figure 28: work6-execution-completion

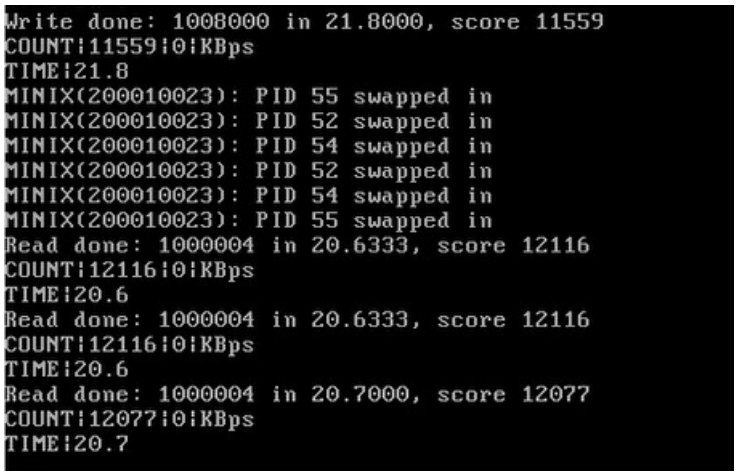

Figure 29: work6-swap