Shriram Ghadge
Roll No. 180010015

CS314 - Lab 3

Part I :

```
315
316  /*=========================================================================*
317   *              schedule_process                 *
318   *=========================================================================*/
319  static int schedule_process(struct schedproc *rmp, unsigned flags)
320  {
321      int err;
322      int new_prio, new_quantum, new_cpu;
323
324      pick_cpu(rmp);
325
326      if (rmp->priority >= 7)
327      {
328          printf("Minix 3: <pid> %d swapped in\n", _ENDPOINT_P(rmp->endpoint));
329      }
330
331      if (flags & SCHEDULE_CHANGE_PRIO)
332          new_prio = rmp->priority;
333      else
334          new_prio = -1;
335
```

Schedule.c

To print whenever a user-level process is brought in by the scheduler,

```
sshd Minix: PID 201 created
PID 177 swapped in
Starting sshd.
Minix: PID 202 created
PID 178 swapped in
Minix: PID 203 created
PID 179 swapped in
Minix: PID 202 exited
Minix: PID 201 exited
 done.
Minix: PID 178 exited
Minix: PID 108 exited
Minix: PID 13 exited
Minix: PID 204 created
PID 180 swapped in
Minix: PID 205 created
PID 181 swapped in
Minix: PID 206 created
PID 182 swapped in
Minix: PID 207 created
PID 183 swapped in

Minix/i386 (192.168.150.187) (console)

login:
```

the code modified in the file:
minix/servers/sched/schedule.c

in the function
schedule_process()

output

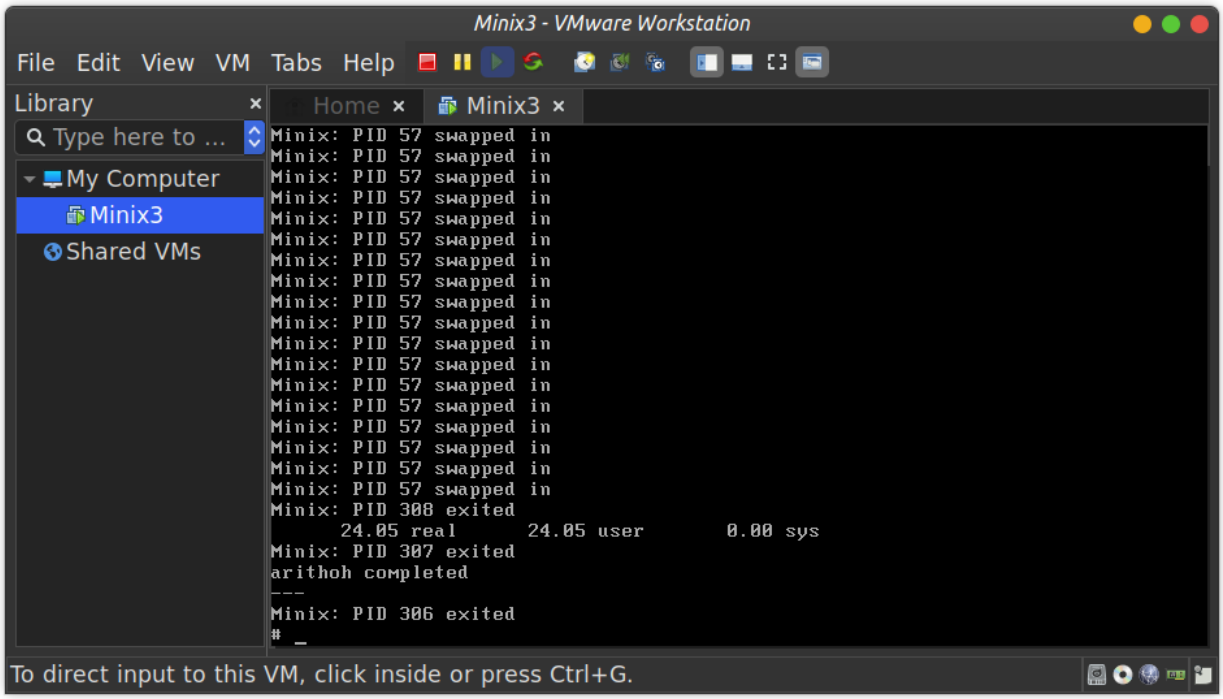
Part II :

- `Arithoh.sh` (CPU Bound Benchmark)

It is observed that while running ‘arithoh’ alone, the real & the user took the same time of (24.05) whereas the sys time taken is 0. The kernel scheduler log conforms with this and shows the message to schedule ./arithoh was sent 92 times consecutively.

Observations after running two executables of ./arithoh parallelly :

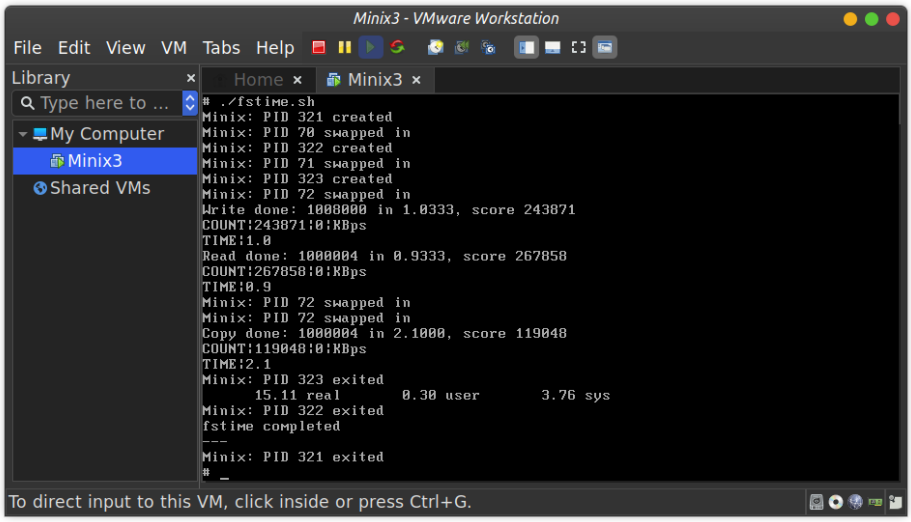
1. ./arithoh.sh & ./arithoh.sh
2. An almost round-robin scheduling in both the kernel scheduler and our print statements. However, the <pid> of one isn’t always followed by the other and is sometimes swapped in again.
3. We also see that although both processes start at the same time, the second one takes around 8s more after the first one, showing that it was given lesser priority during scheduling.



- `Fstime.sh` (IO Bound Benchmark)

  Observations on executing fstime.sh :

  1. total turnaround time >> sys time > user time
  2. This is because the process needs to wait for its IO to complete before continuing.



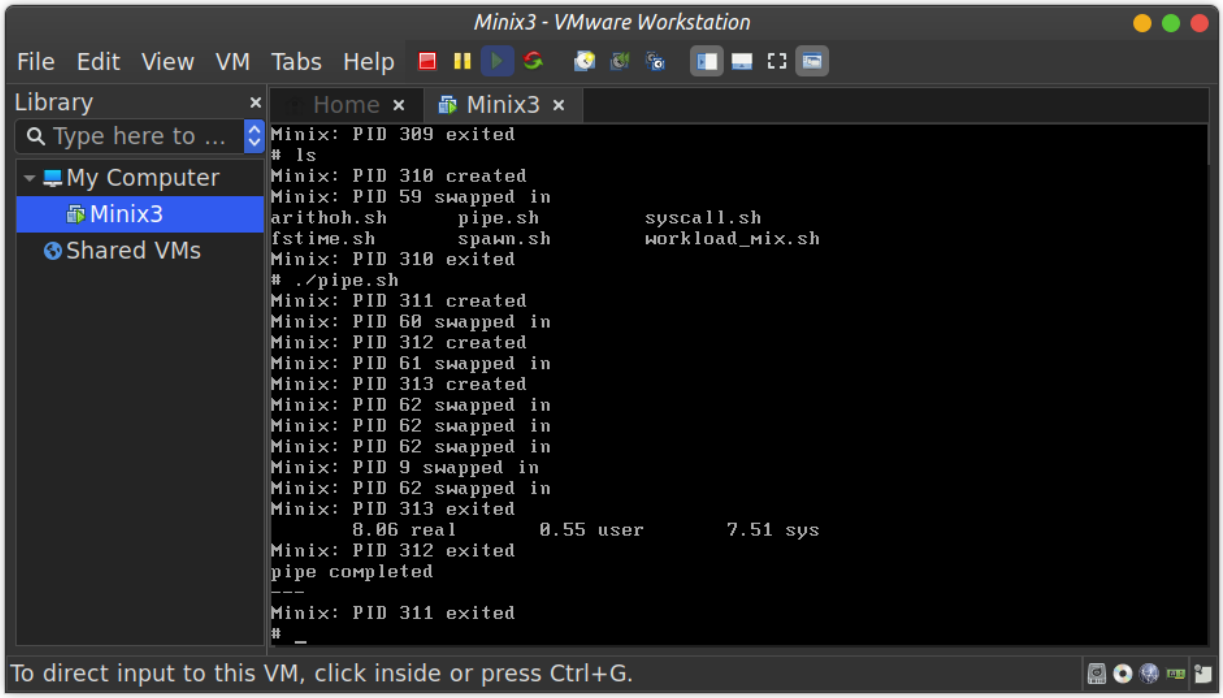
Observations after running ./arithoh and ./fstime at the same time :

1. Arithoh always repeats for some time before IO is scheduled
2. This demonstrates the scheduler's efficiency in utilizing the wait time of ./fstime to schedule a CPU task like ./arithoh.

- `Pipe.sh` (CPU Bound Benchmark)

  Because of Inter-Process Communication protocols, Pipe takes longer amount of time in sys compared to usr

  ./pipe.sh and ./arithoh.sh works similar to ./fstime.sh and ./arithoh.sh, where the pipe finishes earlier followed by consecutive scheduling of arithoh until completion

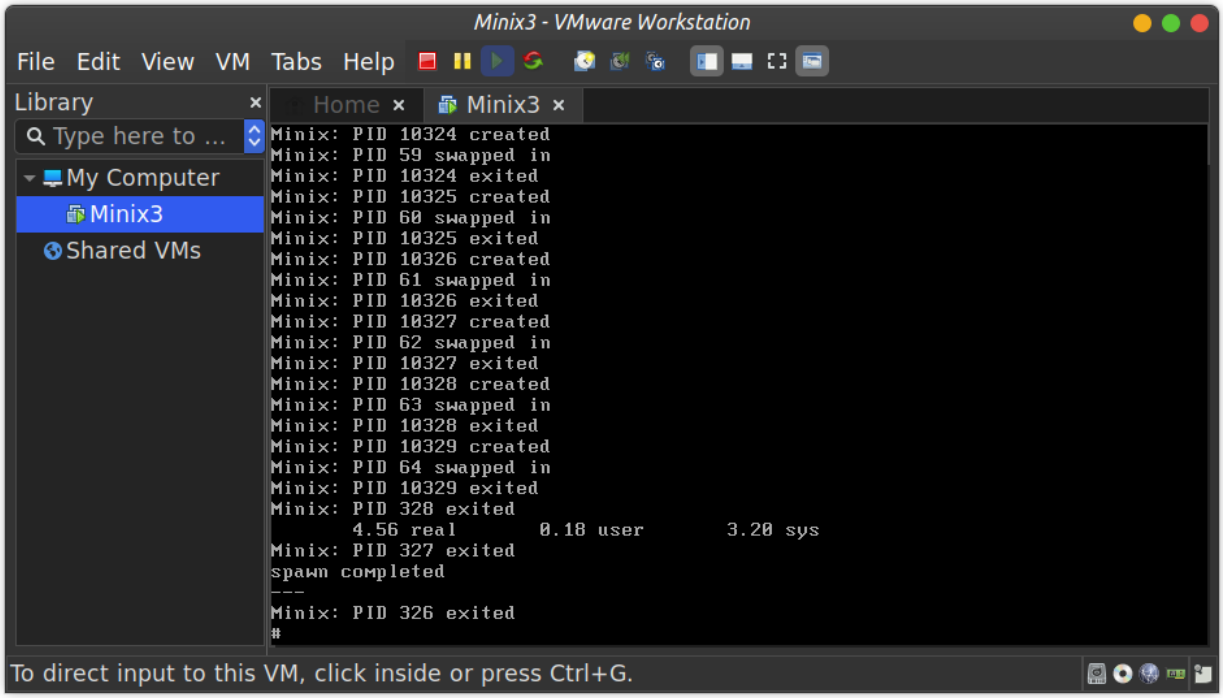


- `Spawn.sh` (CPU Bound Benchmark)

  Similar to pipe, here also it takes longer amount of time in sys compared to usr. We can see that a huge number of processes ranging from 12 to 320 are swapped into the queue consecutively.

  When spawn and arithoh are executed,

  spawn finishes earlier and arithoh continues executing until completion.



- `Syscall.sh` (CPU Bound Benchmark)

    1. Real time > sys time > usr time
    2. ./Syscall and ./arithoh run in a round-robin manner with syscall completing first followed by arithoh.