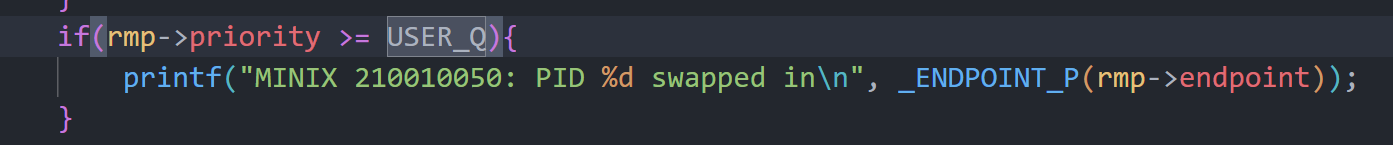
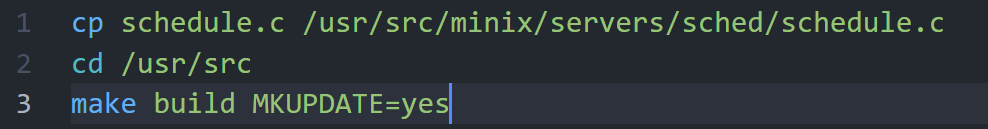
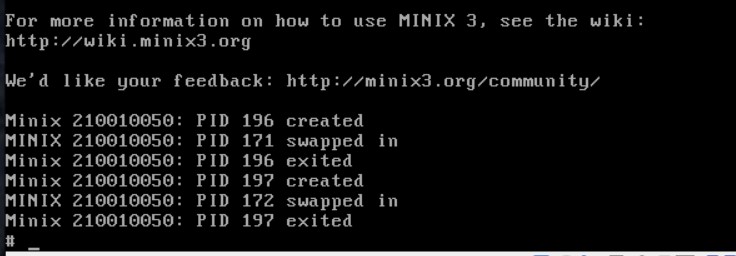
# **­­CS-314 OS LAB-3 REPORT**

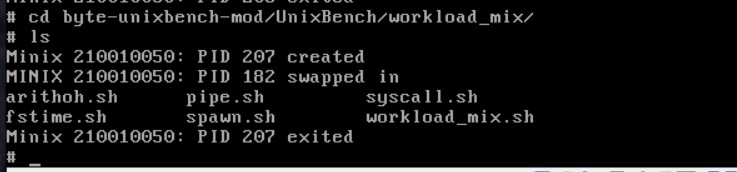
**PART-1**

* To print the string “MINIX <Rollnumber>: PID <pid> swapped in” whenever a user-level process is brought in by the scheduler, we need to modify the source code in the file ‘schedule.c’ at the location: /usr/src/minix/servers/sched  
  We add the following statement at line no. 327 inside the function ‘schedule\_process()’ in a copy of file ‘schedule.c’.
* schedule.c:
* For this to happen, we need to run the bash file shown below, which copies the modified code to the correct location and deploys the changes on our Minix OS.
* run.sh:
* Now, after rebooting the system, we can see the following:



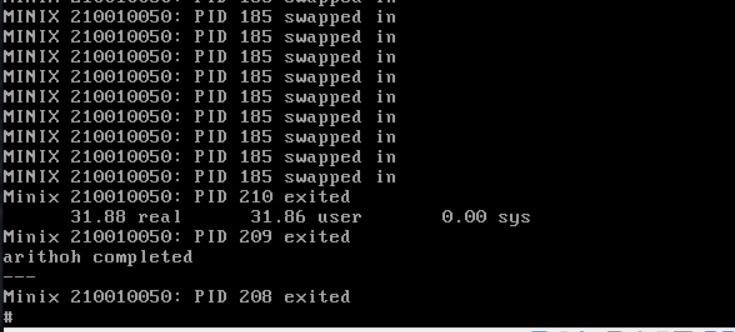
* On running ls command, you can see the following:





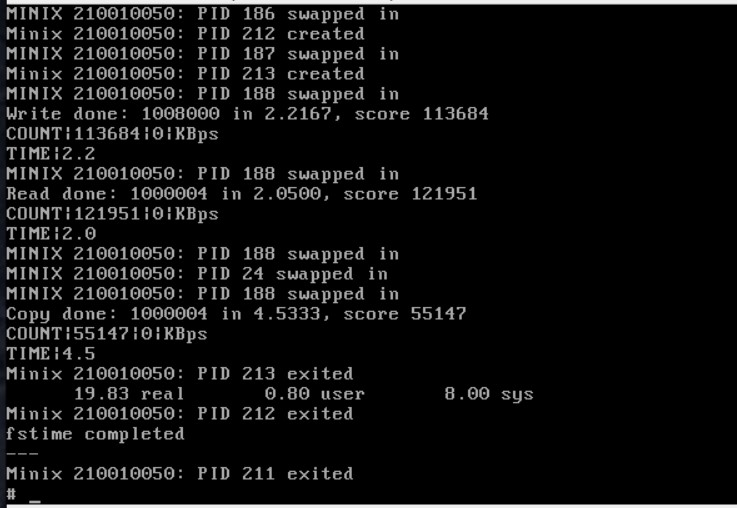
**PART-2**

* We move the source code of the UnixBench benchmark to the home folder in our Minix OS. We then run gmake to build the benchmarks. We then go inside the directory ‘byte-unixbench-mod/UnixBench/workload\_mix’ and run the following benchmarks:
* arithoh.sh:



The arithoh.sh contains arithmetic operations so it is a **CPU Bound** Process since repeated arithmetic operations are rather computationally intensive, and don’t have an I/O component. It is observed that there is no system time used. The entire process is run in user mode as they are cpu intensive processes. Since it is a CPU bound process it has a lower priority hence it’s preemption frequency is high.

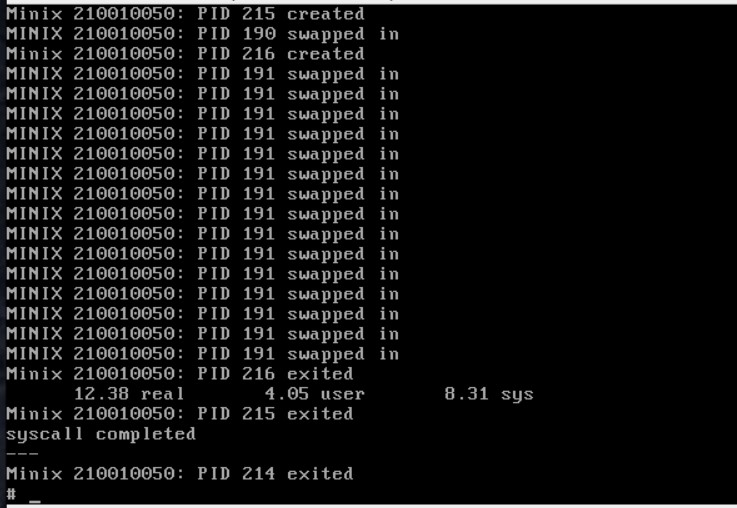
* fstime.sh:



The fstime contains file operations so it is **I/O Bound** Process. Here, the real time is high as compared to user and sys time because the process consists of several calls to time, date, sleep etc.

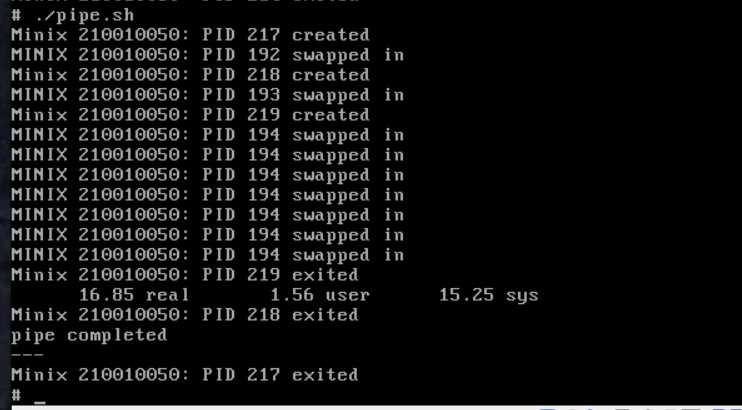
* syscall.sh:

The syscall script seems to mainly consist of **CPU-bound** processes, which does system calls and deals with file descriptors. Hence, we can see that most of the process is run in kernel mode rather than user mode. Their preemption frequency is lower as switching between user and kernel mode has an overhead.



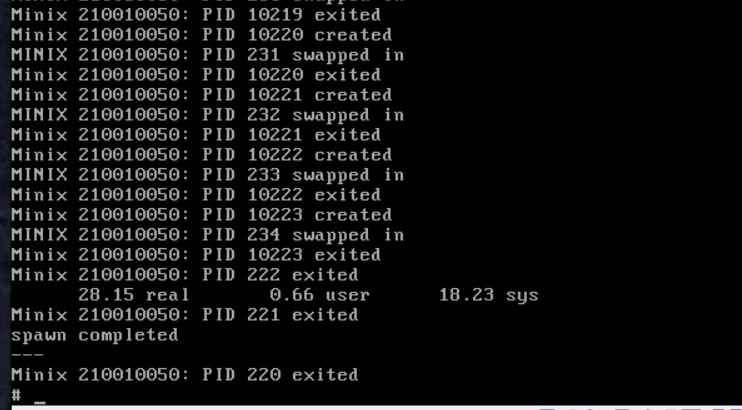
* pipe.sh:

The pipe script seems to be more **I/O-bound**; it involves repeated reads and writes to a pipe. It also has less CPU usage, due to its smaller bursts.

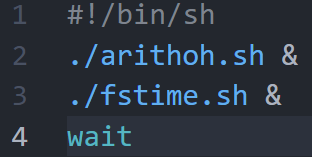


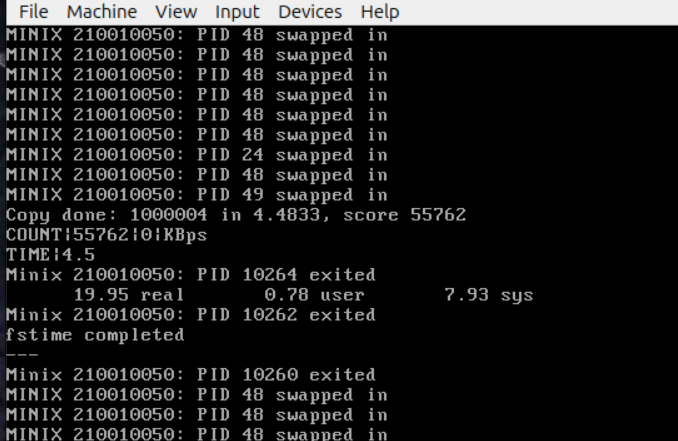
* spawn.sh:

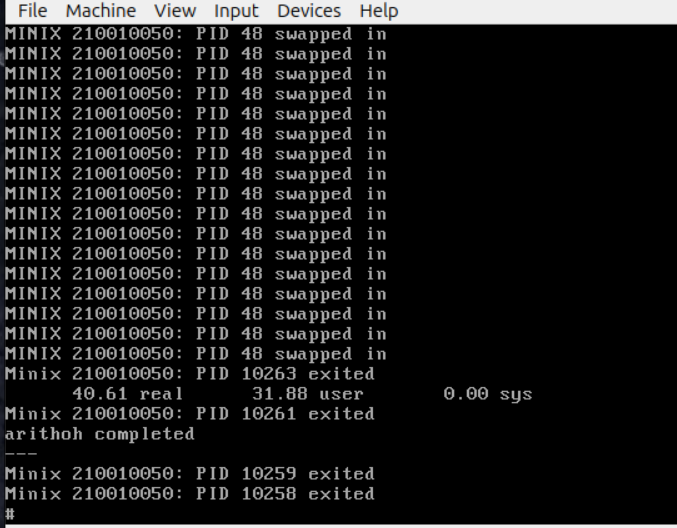
This benchmark consists of system calls and thus seems to be **CPU-bound**. Hence, we can see that most of the process is run in kernel mode rather than user mode. Their preemption frequency is lower as switching between user and kernel mode has an overhead.



* Running a workload\_mix :
  + workload\_mix1.sh:

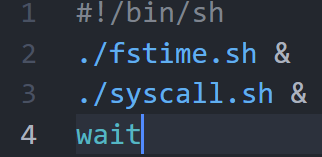


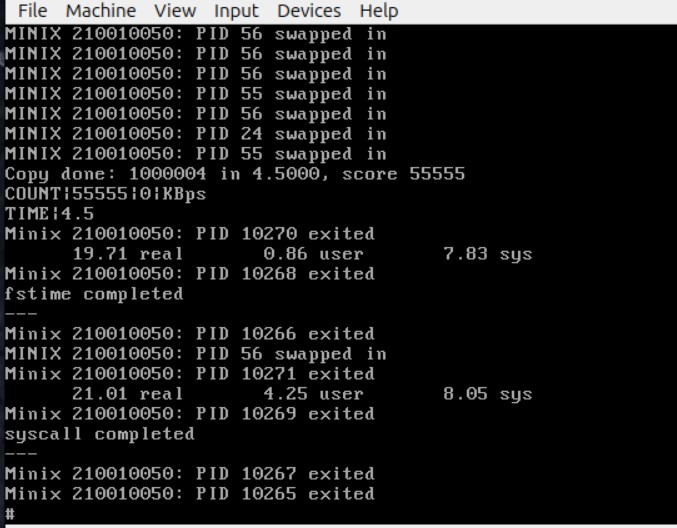




It is observed that, fstime completes its execution before arithoh as fstime mostly consists of real time processes which have the highest priority when compared to arith operations. Also among the two processes, arithoh.sh has a higher pre-emption frequency as it is a lower priority process. We observe that arithoh swaps occur in the time where fstime waits for I/O. The fstime.sh process is switched back to user mode only when all the I/O routines are completed.

* + workload\_mix2.sh:





In this, we run a syscall and an I/O bound process- fstime to observe the process swaps. In this case, the CPU-intensive parts of the syscall code swap in when fstime waits for I/O.