CS 314: Operating Systems Laboratory
Lab 4
Group: 18

Nampally Pranav 190010026
T Satwik 190030043

---

1 Part 1

1.1 Changes to Source Code

The **system.c** file in the minix/kernel directory in the source code acts as an interface between kernel and user level processes. This code, has a function called sched_proc() which takes is takes in the process, assigns quantum values etc. To print the quantum time allotted to a process and the time sued by it, we need to make changes to the **sched_proc()** function. In this function, we can print the quantum size(quantum size allocated to a process is an attribute of the proc structure, which can be found in the kernel/proc.h header file.) Time left(which is also an attribute of a proc structure, but its data type is u64_t and to convert it into ms, we used a function declared in kernel/clock.h) and time used(quantum size-time left). The changes can be seen below.

```
...
int sched_proc(struct proc *p,
                        int priority,
                        int quantum,
                        int cpu)
{
        /* Make sure the values given are within the allowed range.*/
        if ((priority < TASK_Q && priority != -1) || priority > NR_SCHED_QUEUES)
                return(EINVAL);

        if (quantum < 1 && quantum != -1)
                return(EINVAL);
        printf("Quantum  Time Size: %d, Used Quantum Time: %d, Quantum Time Left:%d\n",
        p->p_quantum_size_ms,
        p->p_quantum_size_ms-cpu_time_2_ms(p->p_cpu_time_left),
        cpu_time_2_ms(p->p_cpu_time_left));
#ifdef CONFIG_SMP
        if ((cpu < 0 && cpu != -1) || (cpu > 0 && (unsigned) cpu >= ncpus))
                return(EINVAL);
...
}
```

The screenshot of the os after the build can be seen in Figure 1

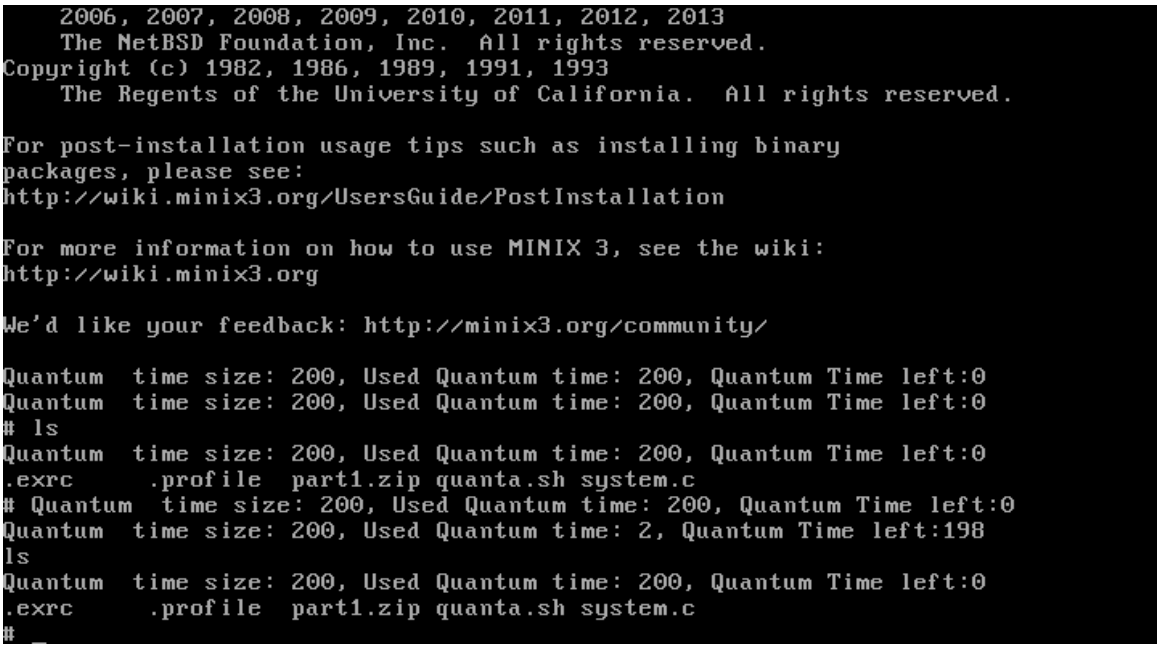


Figure 1: Modified OS after build

1.2 Instructions to run

There are 2 files attached with this report, `system.c` and `system-copy-build.sh`, **Ensure that these files are placed in the /root folder.** Then run the following command

```
$ ./system-copy-build.sh
```

This command will make a copy of the schedule.c file in the appropriate directory and changes the folder /usr/src and builds the OS. After the build is successful perform a reboot, to see the desired output.

Note: To copy files into a directory we need some permissions in the folder. Hence if the shell script shows permission denied then in the /root folder run the following command.

```
$ chmod -R o+rwx root
```

1.3 Workloads and their observations

1.3.1 Workload 1:

The contents of this file are,

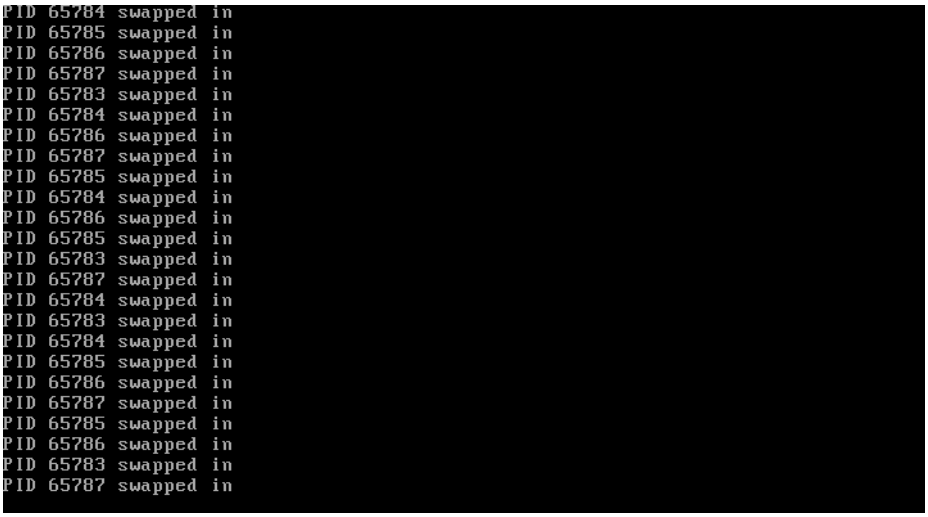
```
#!/bin/sh
./arithoh.sh &
./arithoh.sh &
```

```
./arithoh.sh &
./arithoh.sh &
./arithoh.sh &
wait
```

This workload, executes arithoh.sh task(which is a CPU intensive task), repeatedly for 5 times.

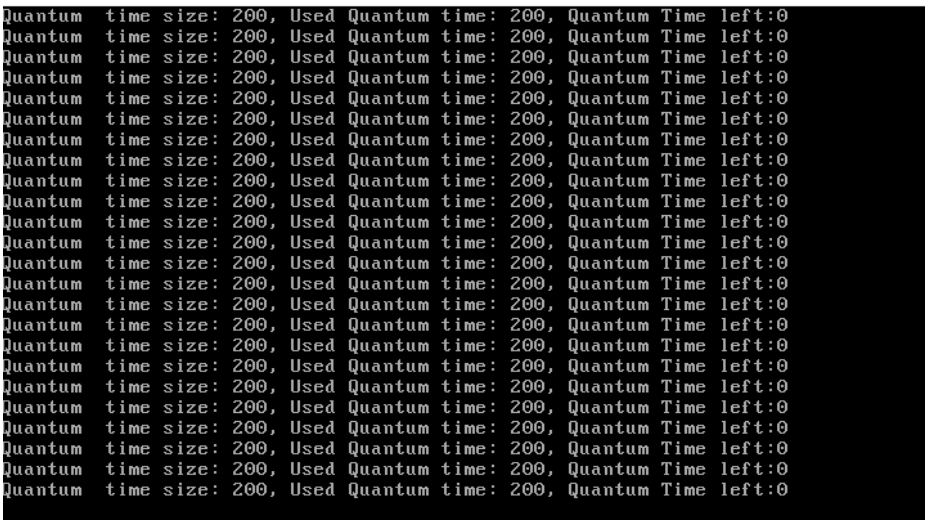
Analysis and Observations:

The following figure shows the output



```
PID 65784 swapped in
PID 65785 swapped in
PID 65786 swapped in
PID 65787 swapped in
PID 65783 swapped in
PID 65784 swapped in
PID 65786 swapped in
PID 65787 swapped in
PID 65785 swapped in
PID 65784 swapped in
PID 65786 swapped in
PID 65785 swapped in
PID 65783 swapped in
PID 65787 swapped in
PID 65784 swapped in
PID 65783 swapped in
PID 65784 swapped in
PID 65785 swapped in
PID 65786 swapped in
PID 65787 swapped in
PID 65785 swapped in
PID 65786 swapped in
PID 65783 swapped in
PID 65787 swapped in
```

Figure 2: workload_mix1 screenshot (quantum)



```
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
```

Figure 3: workload_mix1 screenshot (swapped in)

This workload has, 5 arithoh processed. arithoh is a CPU intensive task, and needs more CPU

time consistently.
As we can see from the screenshots, processes, ...83, ...84, ...85, ...86 and ...87 are the 5 arithoh processes. As we can see in the screenshot the 5 processes are executed alternatively, and hence each of them are swapped in the same order. Then the arithoh processes are also exited in the same order.

Time quanta allotted for these processes is:200 ms, and it uses the whole time slice, because it is CPU intensive process.
So, from this workload we can understand that the processor tries to give equal priority to same kind of tasks. Here all the 3 processes are equivalent, and hence they get equal share of CPU(they are executed in round-robin fashion). Though this process is fair, and treats these processes equally, it also means that the number of context switches are more.

1.3.2 Workload 2:

The contents of this file are,

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

This workload, executes all the mentioned programs.

Analysis and Observations:
The following figure shows the output

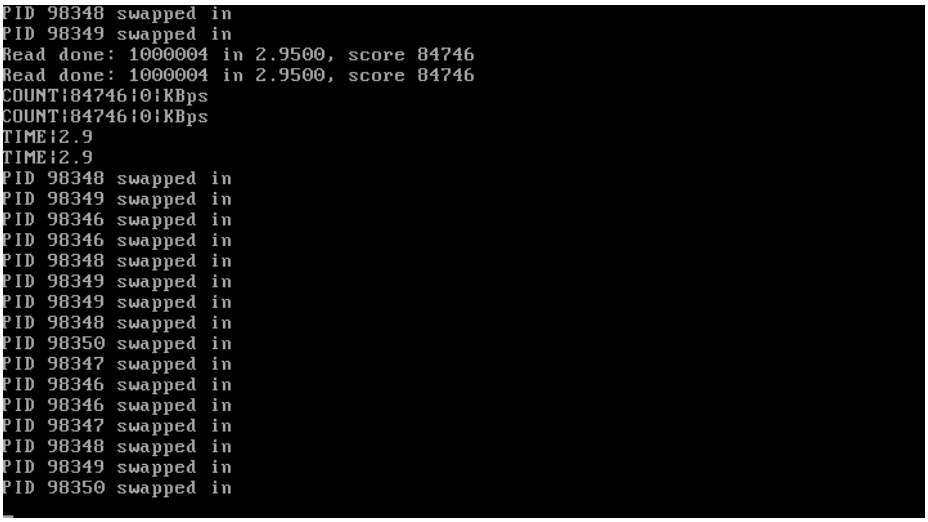
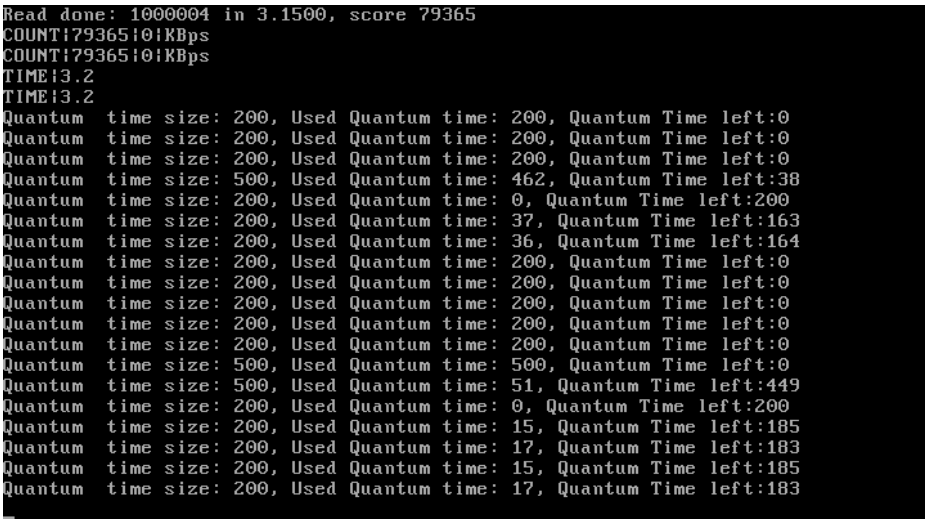


Figure 4: workload_mix2 screenshot (quantum)



```
Read done: 1000004 in 3.1500, score 79365
COUNT|79365|0|KBps
COUNT|79365|0|KBps
TIME|3.2
TIME|3.2
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 500, Used Quantum time: 462, Quantum Time left:38
Quantum  time size: 200, Used Quantum time: 0, Quantum Time left:200
Quantum  time size: 200, Used Quantum time: 37, Quantum Time left:163
Quantum  time size: 200, Used Quantum time: 36, Quantum Time left:164
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 500, Used Quantum time: 500, Quantum Time left:0
Quantum  time size: 500, Used Quantum time: 51, Quantum Time left:449
Quantum  time size: 200, Used Quantum time: 0, Quantum Time left:200
Quantum  time size: 200, Used Quantum time: 15, Quantum Time left:185
Quantum  time size: 200, Used Quantum time: 17, Quantum Time left:183
Quantum  time size: 200, Used Quantum time: 15, Quantum Time left:185
Quantum  time size: 200, Used Quantum time: 17, Quantum Time left:183
```

Figure 5: workload_mix2 screenshot (swapped in)

This workload has, 2 aritoh processes and 2 fstime processes and 1 syscall process. aritoh and syscall is a CPU intensive task, and needs more CPU time consistently, and fstime is I/O based operation.
As we can see from the screenshots, processes, ...48, ...49 are syscall processes and ...47, ...50 are fstime processes and ...46 is an aritoh process. First the fstime process is run, but it doesn't utilize the whole time slice, because it waits for an I/O in the mean time a syscall process or aritoh process is scheduled.

Time quanta alloted for sycall and aritoh (CPU intensive) processes is:200 ms, and they tend to use the whole time slice, and for fstime process, the time quanta alloted (I/O intensive) processes is:500 ms, and they don't use the whole time slice as they wait for tasks as we can see in the above screenshot.
So, from this workload we can learn that the scheduler runs the I/O operations for only a few time slices and schedules different processes, while the I/O operations are made. This leads to a better utilization of CPU.

1.3.3 Workload 3:

The contents of this file are,

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

This workload, exhas aritoh and fstime processes alternatively.

Analysis and Observations: This workload has, 3 aritoh processes and 2 fstime processes which are given alternatively.

Analysis and Observations: The following figure shows the output

```
PID 98363 swapped in
PID 98365 swapped in
PID 98367 swapped in
PID 98363 swapped in
PID 98365 swapped in
PID 98367 swapped in
PID 98363 swapped in
PID 98365 swapped in
PID 98367 swapped in
PID 98363 swapped in
PID 98365 swapped in
PID 98367 swapped in
PID 98363 swapped in
PID 98365 swapped in
PID 98367 swapped in
PID 98363 swapped in
PID 98365 swapped in
PID 98367 swapped in
PID 98363 swapped in
PID 32859 swapped in
PID 98363 swapped in
PID 98365 swapped in
PID 98367 swapped in
PID 32859 swapped in
```

Figure 6: workload_mix3 screenshot (quantum)

```
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 500, Used Quantum time: 500, Quantum Time left:0
Read done: 1000004 in 2.1000, score 119048
Read done: 1000004 in 2.1000, score 119048
COUNT|119048|0|KBps
COUNT|119048|0|KBps
TIME|2.1
TIME|2.1
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 500, Used Quantum time: 392, Quantum Time left:108
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 200, Used Quantum time: 0, Quantum Time left:200
Quantum  time size: 200, Used Quantum time: 73, Quantum Time left:127
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
```

Figure 7: workload_mix3 screenshot (swapped in)

As we can see from the screenshots, processes, ...63, ...65 and ...67 are aritoh processes and ...64, ...66 are fstime processes. The aritoh processes are run and in between the fstime processes are run, however they do not utilize the time slices completely and hence aritoh processes are run while, fstime waits for input.

Time quanta alloted for sycall and aritoh (CPU intensive) processes is:200 ms, and they tend to use the whole time slice, and for fstime process, the time quanta alloted (I/O intensive) processes is:500 ms, and they don't use the whole time slice as they wait for tasks as we can see in the above

screenshot.

1.3.4 Workload 4:

The contents of this file are,

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

This workload, executes fstime.sh task(which is a I/O intensive task), repeatedly for 5 times.

Analysis and Observations: The following figure shows the output

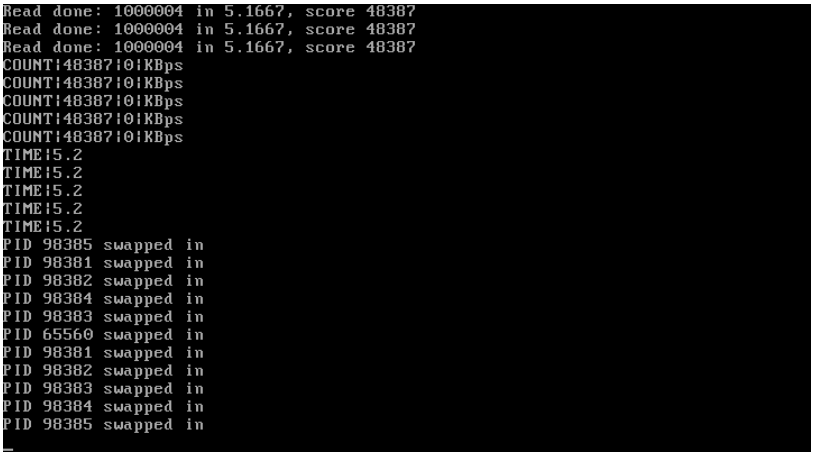


Figure 8: workload_mix4 screenshot (quantum)

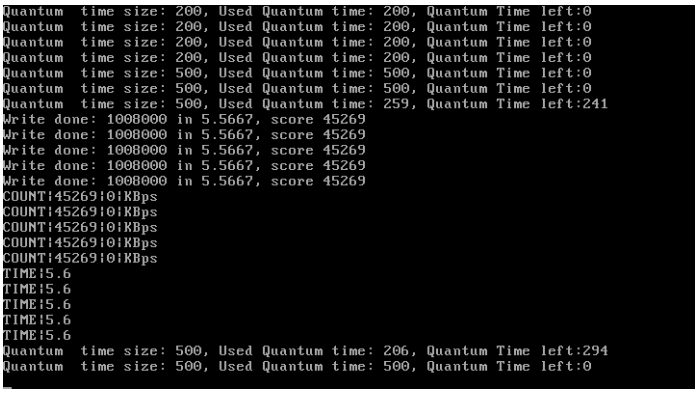


Figure 9: workload_mix4 screenshot (swapped in)

This workload has, 5 fstime processes. Fstime is an I/O intensive task, and needs less CPU time, as they wait for input.
As we can see from the screenshots, processes, ...81, ...82, ...83, ...84, ...85 are the 5 fstime processes. As we can see in the screenshot the 5 processes are swapped in before a read or write corresponding to a process is done.

Time quanta allotted for these processes is:500 ms, and they do not use the whole time slice, because it is I/O intensive process.

2 Part 2

2.1 Changes to Source Code

The **schedule.c** file in the minix/servers/sched directory in the source code acts contains the scheduling policy. This code, by default follows the Round-Robin scheduling within a queue, but we need to update the policy to be Pseudo-FIFO policy, which basically means that each time a process has to be selected the scheduler has to select the oldest process. This also means that when a process uses the CPU, it is the oldest one, and hence a different process may enter only after it is executed. To change the scheme, we can make the following changes in this schedule.c file.

The round robin code, was implemented by incrementing the priority(numerically) each time a process is alloted a quantum/time slice with the CPU, in the **do_noquantum()** function, hence when we pick the next process, the increase in priority ensures that the next process is picked, in the next time slice. However, if we decrement the priority(numerically), then each time the same process is picked, until it is done. Hence the oldest process is alloted time slice first then it has the CPU until it completes execution.

In balance_queues() fucntion, there is an extra decrement condition, which basically checks that priority doesn't exceed the max value, however in our case we are decrementing the priority, and hence we can remove this line, and also we need to check that the priority doesn't surpass MAX_USER_Q.

```
...
int do_noquantum(message *m_ptr)
{
    ...
        rmp = &schedproc[proc_nr_n];
        if (rmp->priority < MIN_USER_Q) {
                // rmp->priority += 1; /* lower priority Replace this line with the below one */
                rmp->priority -= 1;
        }
    ...
}
static void balance_queues(minix_timer_t *tp)
{
    ...
        for (proc_nr=0, rmp=schedproc; proc_nr < NR_PROCS; proc_nr++, rmp++) {
                if (rmp->flags & IN_USE) {
                        if (rmp->priority > rmp->max_priority) {
                                // rmp->priority -= 1; /* remove this line by commenting */
```

```
                                schedule_process_local(rmp);
                        }
                }
        }
    ...
}
...
```

2.2 Instructions to run

There are 2 files attached with this report, `schedule.c` and `schedule-copy-build.sh`, **Ensure that these files are placed in the /root folder.** Then run the following command

```
$ ./system-copy-build.sh
```

This command will make a copy of the schedule.c file in the appropriate directory and changes the folder /usr/src and builds the OS. After the build is successful perform a reboot, to see the desired output.

Note: To copy files into a directory we need some permissions in the folder. Hence if the shell script shows permission denied then in the /root folder run the following command.

```
$ chmod -R o+rwx root
```

2.3 Workloads and their observations

2.3.1 Workload 1:

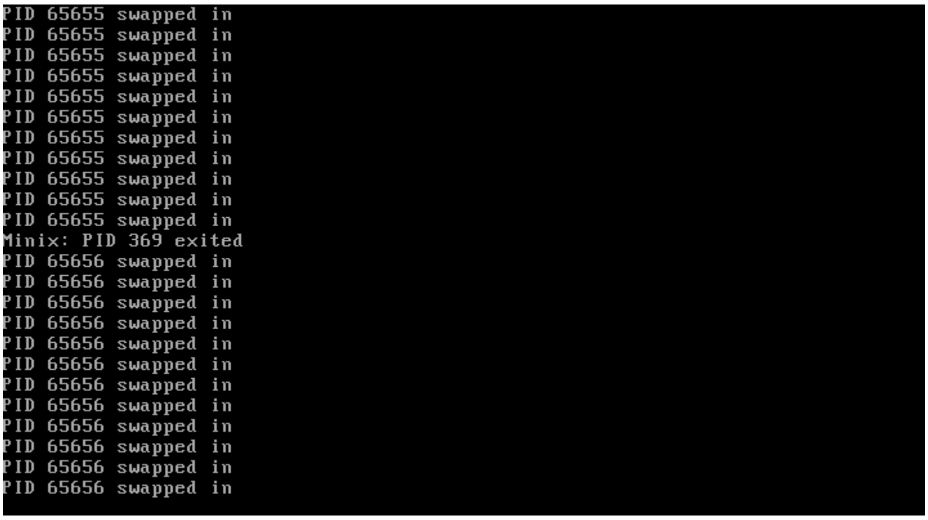
The contents of this file are,

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

This workload, executes arithoh.sh task(which is a CPU intensive task), repeatedly for 5 times.

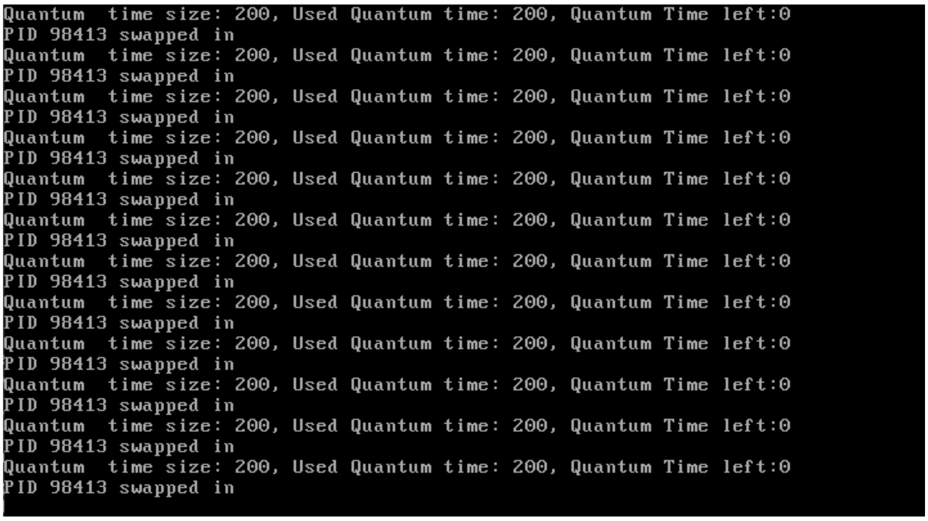
Analysis and Observations:

The following figure shows the output



```
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
PID 65655 swapped in
Minix: PID 369 exited
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
PID 65656 swapped in
```

Figure 10: workload_mix1 screenshot (quantum)



```
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 98413 swapped in
```

Figure 11: workload_mix1 screenshot (swapped in)

This workload has, 5 arithoh processed. arithoh is a CPU intensive task, and needs more CPU time consistently.
As we can see from the screenshots, processes, ...54, ...55, ...56, ...57 and ...58 are the 5 arithoh processes. As we can see in the screenshot each process is continuously executed, i.e ...54 is executed until it is completely executed and then 55 is done and so on.

Time quanta allotted for these processes is:200 ms, and it uses the whole time slice, because it is CPU intensive process. So, from this workload we can understand that the processor is executed one process until it completed and though all the 5 processes are of same nature and have same priority, we give more preference to the process that came first, and hence the oldest process is picked at

each stage, representing the required FIFO strategy.

2.3.2 Workload 2:

The contents of this file are,

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

This workload, executes all the mentioned programs.

Analysis and Observations:

The following figure shows the output

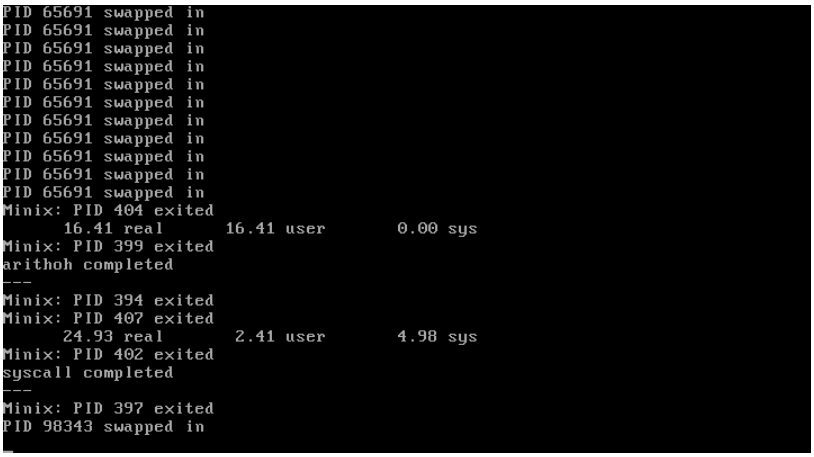


Figure 12: workload_mix2 screenshot (quantum)

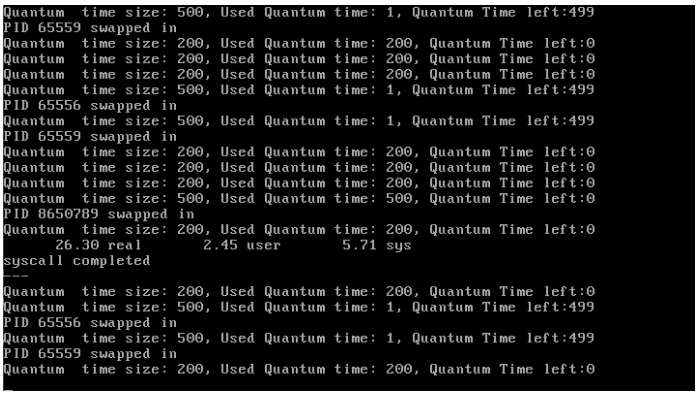


Figure 13: workload_mix2 screenshot (swapped in)

This workload has, 1 aritoh processes and 2 fstime processes and 2 syscall process. arithoh and syscall is a CPU intensive task, and needs more CPU time consistently, and fstime is I/O based operation.
As we can see from the screenshots, processes, ...93, ...94 are syscall processes and ...92, ...95 are fstime processes and ...91 is an aritoh process.

Here the arithoh process is executed first and since it is CPU intensive process, it is continuously swapped in until the execution is complete. However, the fstime process is an I/O intensive process and it is sent to the blocked/wait queue while it waits for an input. So when we use the FIFO policy, it picks from the runnable/ready queue and hence the next oldest process syscall is executed, and it being a CPU intensive process executes until completion and hence it is completed before the fstime process.

Time quanta alloted for syscall and arithoh (CPU intensive) processes is:200 ms, and they tend to use the whole time slice, and for fstime process, the time quanta alloted (I/O intensive) processes is:500 ms, and they don't use the whole time slice as they wait for tasks as we can see in the above screenshot.

2.3.3 Workload 3:

The contents of this file are,

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

This workload has aritoh and fstime processes alternatively. This workload has, 3 aritoh processes and 2 fstime processes which are given alternatively.
Analysis and Observations:
The following figure shows the output

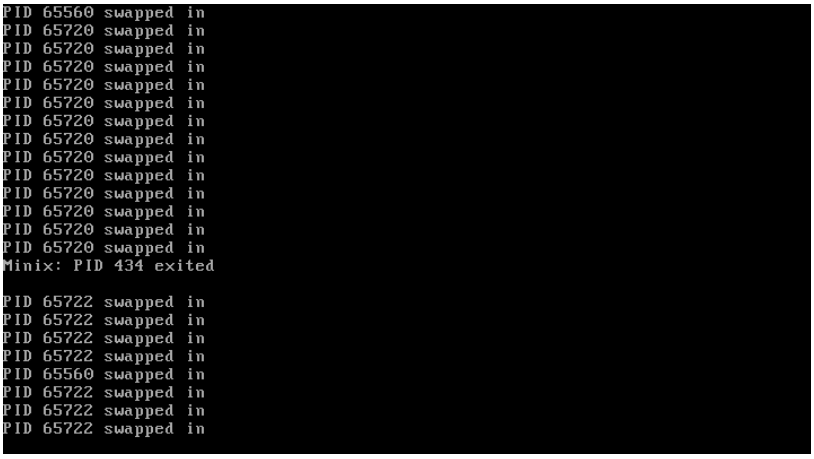
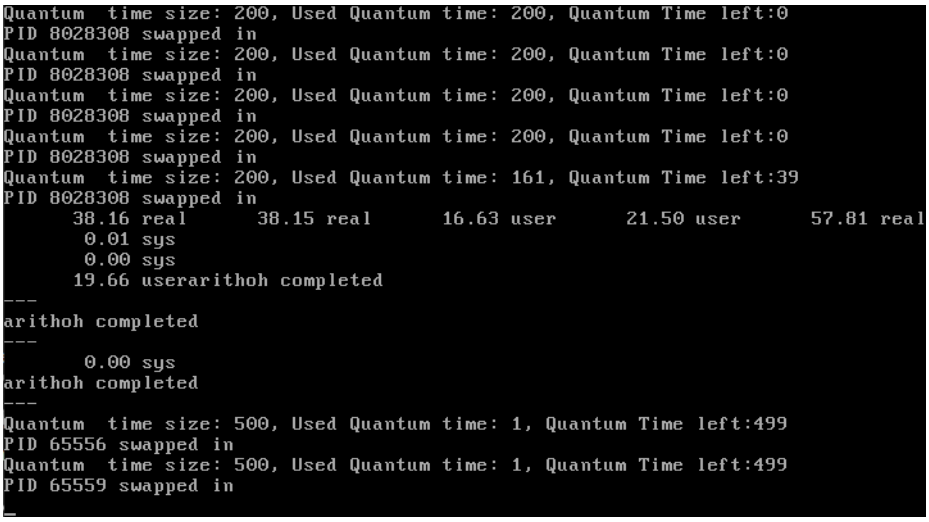


Figure 14: workload_mix3 screenshot (quantum)



```
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 8028308 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 8028308 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 8028308 swapped in
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
PID 8028308 swapped in
Quantum  time size: 200, Used Quantum time: 161, Quantum Time left:39
PID 8028308 swapped in
      38.16 real      38.15 real      16.63 user      21.50 user      57.81 real
       0.01 sys
       0.00 sys
      19.66 userarithoh completed
---
arithoh completed
---
       0.00 sys
arithoh completed
---
Quantum  time size: 500, Used Quantum time: 1, Quantum Time left:499
PID 65556 swapped in
Quantum  time size: 500, Used Quantum time: 1, Quantum Time left:499
PID 65559 swapped in
```

Figure 15: workload_mix3 screenshot (swapped in)

This workload has, 3 aritoh processes and 2 fstime processes. aritoh is a CPU intensive task, and needs more CPU time consistently, and fstime is I/O based operation.

As we can see from the screenshots, processes, ...18, ...20 and ...22 are aritoh processes and ...19, ...21 are fstime processes.

The aritoh processes is first executed, and then the fstime process is run, since it gets moved to the blocked queue, the second aritoh process is executed until completion, and then the fstime is again run, and it is still waiting, so the next aritoh process is also bought in. So the 3 aritoh process are completed first and then the 2 fstime processes are completed.

Time quanta alloted for aritoh (CPU intensive) processes is:200 ms, and they tend to use the whole time slice, and for fstime process, the time quanta alloted (I/O intensive) processes is:500 ms, and they don't use the whole time slice as they wait for tasks as we can see in the above screenshot.

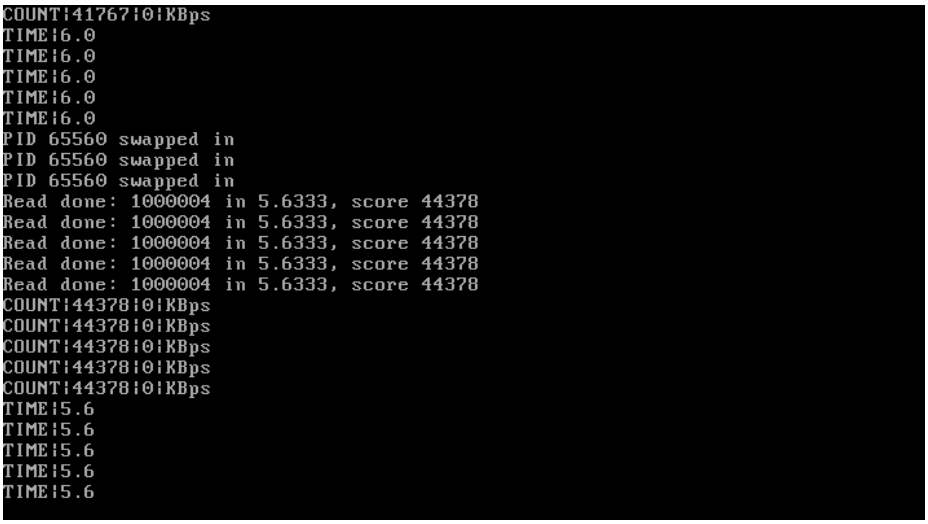
2.3.4 Workload 4:

The contents of this file are,

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

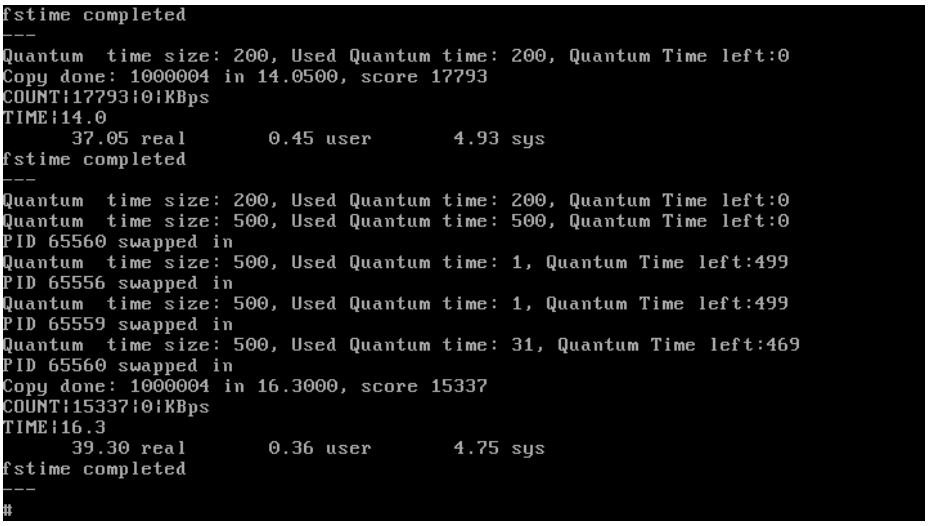
This workload, executes fstime.sh task(which is a I/O intensive task), repeatedly for 5 times.

Analysis and Observations: The following figure shows the output



```
COUNT¦41767¦0¦KBps
TIME¦6.0
TIME¦6.0
TIME¦6.0
TIME¦6.0
TIME¦6.0
PID 65560 swapped in
PID 65560 swapped in
PID 65560 swapped in
Read done: 1000004 in 5.6333, score 44378
Read done: 1000004 in 5.6333, score 44378
Read done: 1000004 in 5.6333, score 44378
Read done: 1000004 in 5.6333, score 44378
Read done: 1000004 in 5.6333, score 44378
COUNT¦44378¦0¦KBps
COUNT¦44378¦0¦KBps
COUNT¦44378¦0¦KBps
COUNT¦44378¦0¦KBps
COUNT¦44378¦0¦KBps
TIME¦5.6
TIME¦5.6
TIME¦5.6
TIME¦5.6
TIME¦5.6
```

Figure 16: workload_mix4 screenshot (quantum)



```
fstime completed
---
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Copy done: 1000004 in 14.0500, score 17793
COUNT¦17793¦0¦KBps
TIME¦14.0
      37.05 real       0.45 user       4.93 sys
fstime completed
---
Quantum  time size: 200, Used Quantum time: 200, Quantum Time left:0
Quantum  time size: 500, Used Quantum time: 500, Quantum Time left:0
PID 65560 swapped in
Quantum  time size: 500, Used Quantum time: 1, Quantum Time left:499
PID 65556 swapped in
Quantum  time size: 500, Used Quantum time: 1, Quantum Time left:499
PID 65559 swapped in
Quantum  time size: 500, Used Quantum time: 31, Quantum Time left:469
PID 65560 swapped in
Copy done: 1000004 in 16.3000, score 15337
COUNT¦15337¦0¦KBps
TIME¦16.3
      39.30 real       0.36 user       4.75 sys
fstime completed
---
#
```

Figure 17: workload_mix4 screenshot (swapped in)

This workload has, 5 fstime processes. fstime is an I/O intensive task, and needs less CPU time, as they wait for input.
As we can see from the screenshots, processes, ...69, ...70, ...71, ...72, ...73 are the 5 fstime processes.

Each fstime process is sent to the blocked queue, while it waits the input, so the other fstime process are executed, and hence the order of execution and time taken isn't really different from the earlier case.

Time quanta allotted for these processes is:500 ms, and they do not use the whole time slice, because it is I/O intensive process.