Operating Systems Lab

# Lab – 5

## Rahul Cheryala, 210010012

# **PART – 1**

In this assignment we are asked to implement at least 4 workload mixes having different characteristics, ranging from all compute-intensive benchmarks to all I/O-intensive benchmarks. Each workload contains around 5 processes in it. We have changed the system.c file for displaying the time slice that has been fixed for a process and time slice used by the process.

1. Workload\_mix1.sh

!/bin/sh

./arithoh.sh &

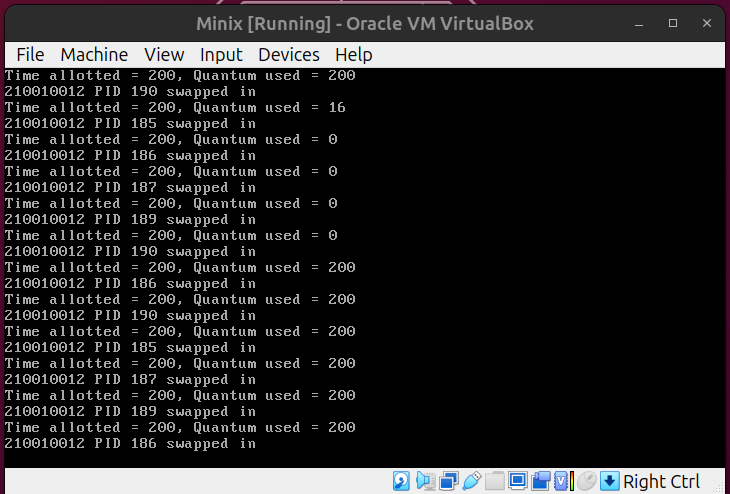
./arithoh.sh &

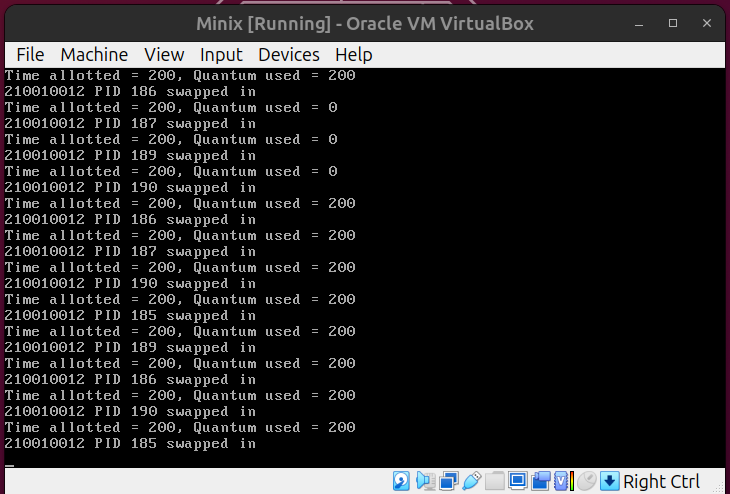
./arithoh.sh &

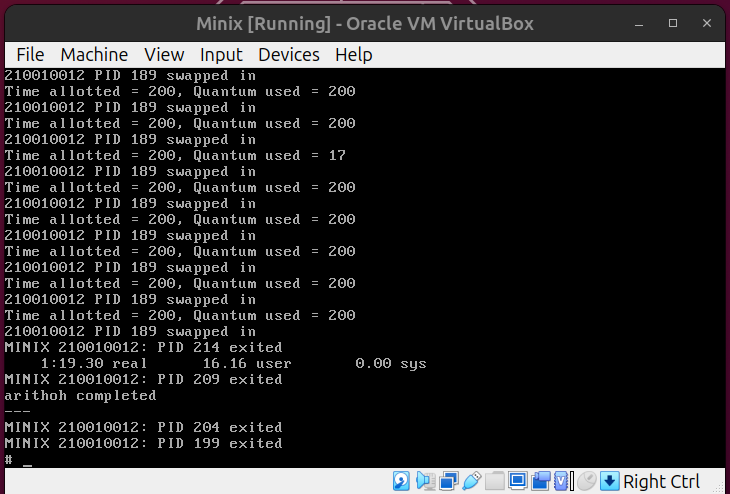
./arithoh.sh &

./arithoh.sh &

Wait







Multiple instances of the CPU-intensive "arithoh.sh" script run concurrently using the default Round-Robin scheduling. Each process is allotted a 200-millisecond time slice, which they fully utilize due to their CPU-bound nature. Consequently, all processes finish their tasks nearly simultaneously, maximizing CPU resource utilization.

2. Workload\_mix2.sh

!/bin/sh

./arithoh.sh &

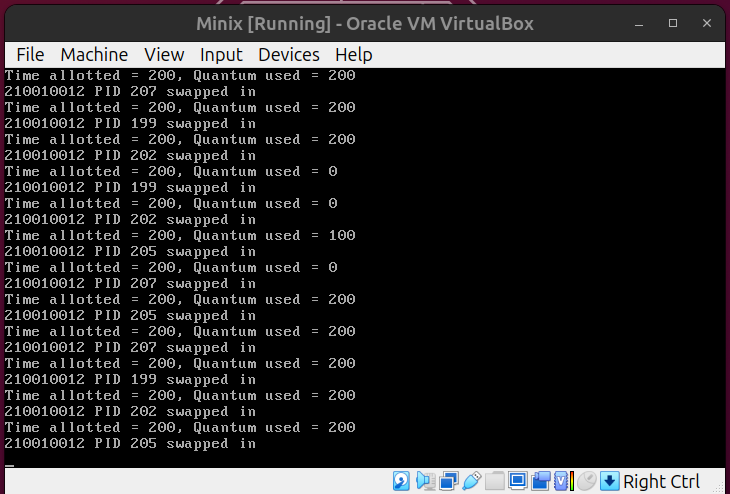
./arithoh.sh &

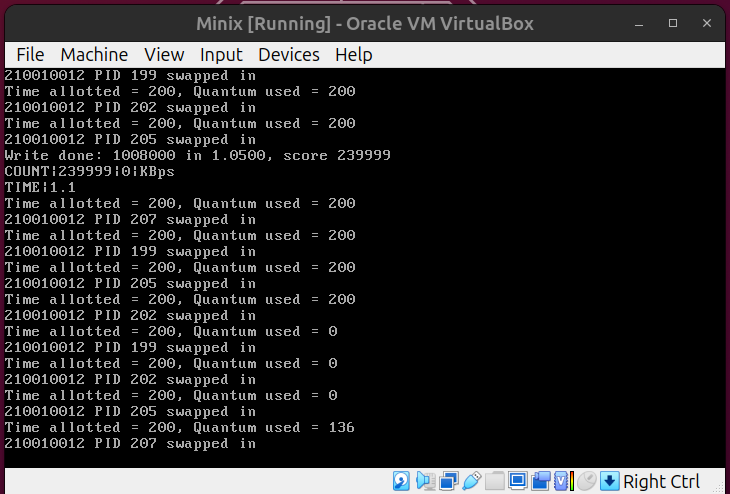
./arithoh.sh &

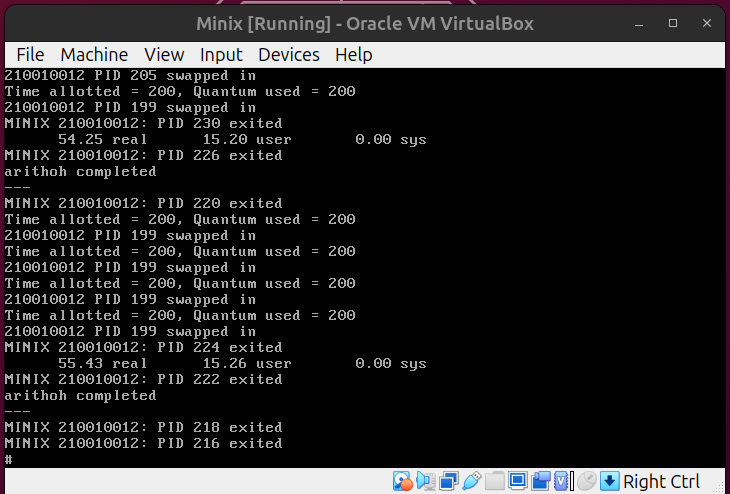
./fstime.sh &

./syscall.sh &

wait







In this scenario, multiple CPU-intensive "arithoh.sh" and one "syscall.sh" process execute in Round Robin fashion with a time slice of 200ms. Additionally, an I/O-intensive "fstime.sh" process is allotted a longer time slice of 500ms but doesn't always use it fully due to waiting for I/O. After CPU-bound tasks finish, remaining time is shared between "arithoh.sh" processes. CPU-intensive tasks take longer, while I/O-bound tasks wait for I/O completion.

3. Workload\_mix3.sh

!/bin/sh

./syscall.sh &

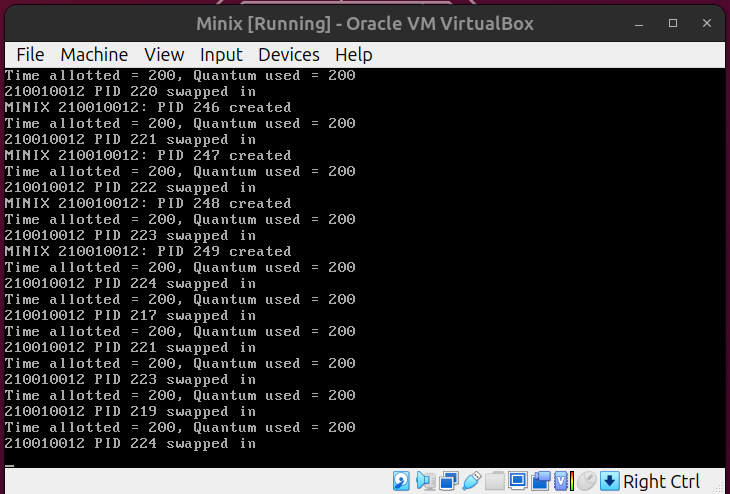
./syscall.sh &

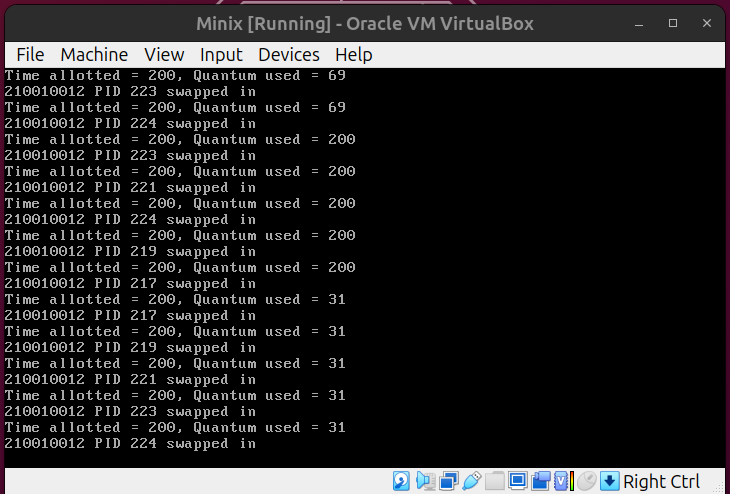
./syscall.sh &

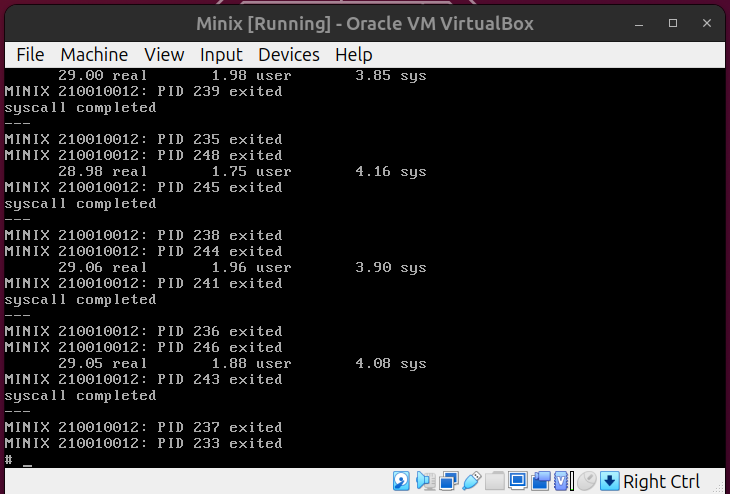
./syscall.sh &

./syscall.sh &

wait







All five processes are "syscall.sh" executing concurrently in Minix3 using Round-Robin scheduling with a time slice of 200. Unlike CPU-intensive tasks, "syscall.sh" processes don't fully engage every time they're scheduled due to their lower intensity. Since they lack I/O operations, they utilize the complete time slice. Consequently, all processes finish nearly simultaneously.

4. Workload\_mix4.sh

!/bin/sh

./fstime.sh &

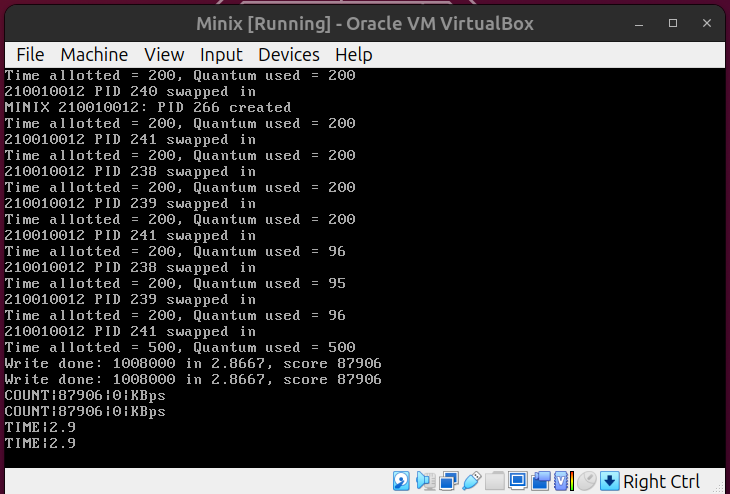
./fstime.sh &

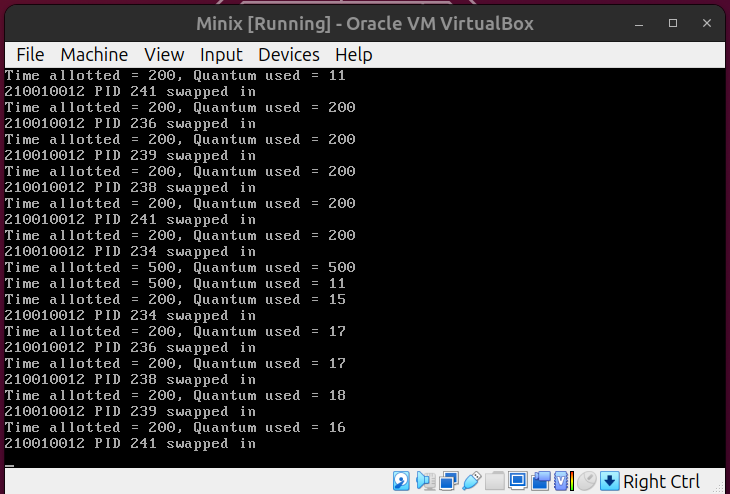
./syscall.sh &

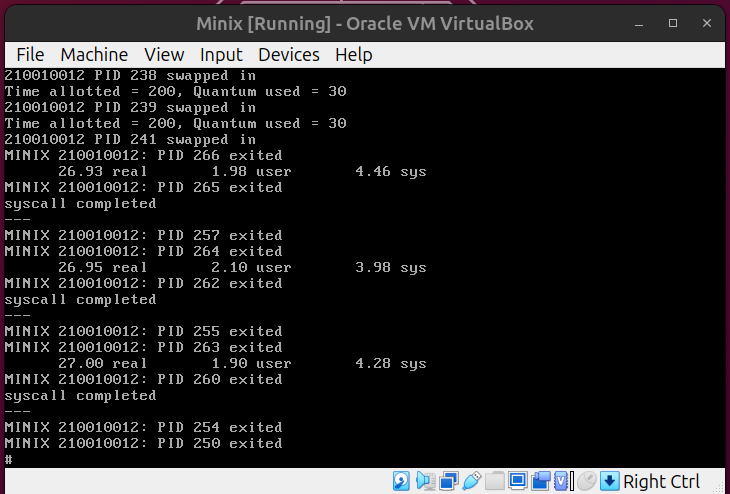
./syscall.sh &

./syscall.sh &

wait







In this scenario, two processes are I/O-intensive "fstime.sh," while three are CPU-intensive "syscall.sh" executing in Round Robin fashion. "fstime.sh" is allocated a longer time slice of 500ms but doesn't always use it fully due to I/O waits. CPU-bound processes take longer, while I/O-bound processes wait for I/O completion.

5. Workload\_mix5.sh

!/bin/sh

./fstime.sh &

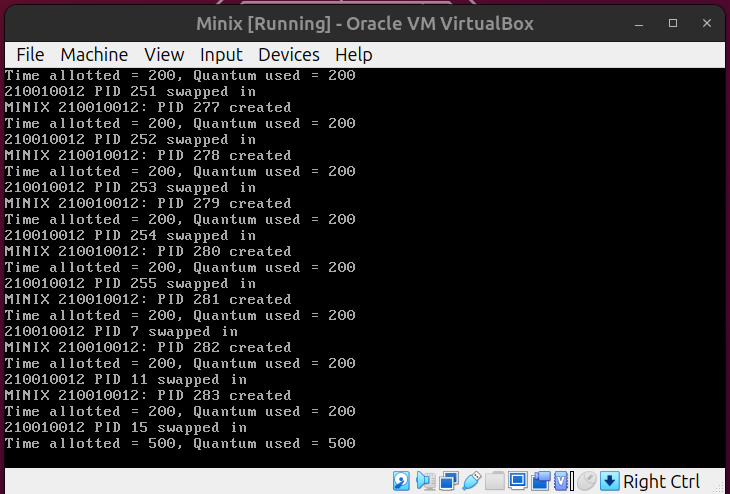
./fstime.sh &

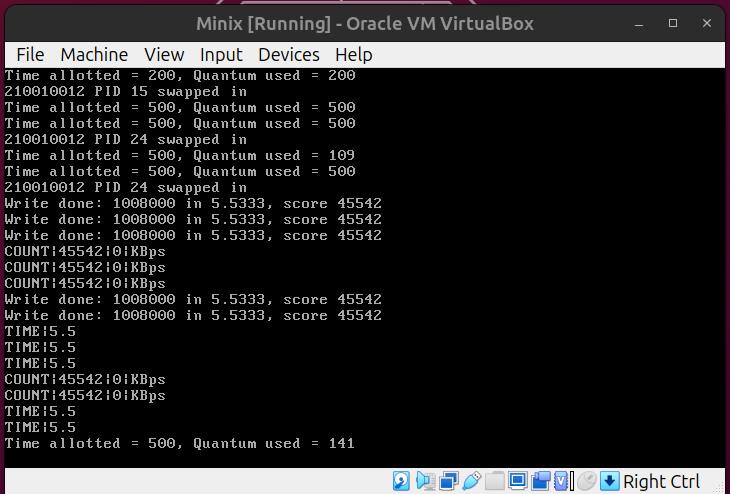
./fstime.sh &

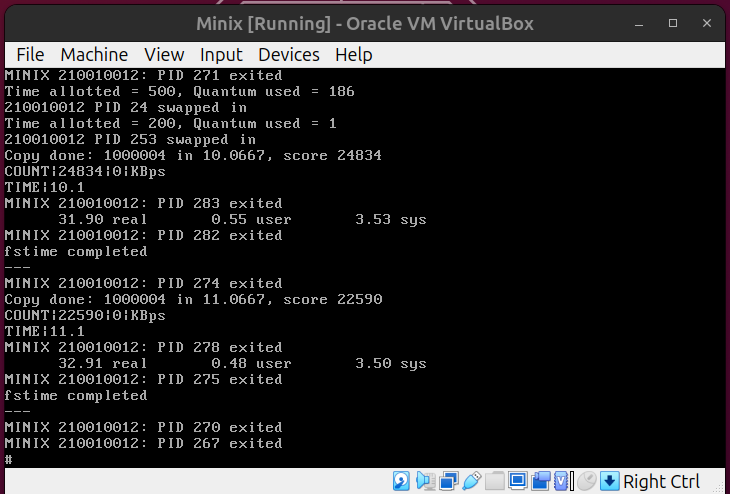
./fstime.sh &

./fstime.sh &

wait







All five processes are I/O-intensive "fstime.sh" in workload mix5.sh. They execute sequentially in Round-Robin fashion, waiting for I/O before CPU execution. As I/O-bound tasks, they don't utilize their full time slice, waiting for I/O completion before proceeding. All processes wait for I/O and then complete operations in order.

# **PART – 2**

In 2nd part, we have to modify the user-level scheduler in Minix3 to the following “PseudoFIFO” policy: among the user-level processes that are ready to execute, the one that entered the earliest must be scheduled. Here Minix3 by follows Round Robin Scheduling by default and we have changed it to Pseudo FIFO as following:

In function do noquantum(), to prioritize in the queue we have changed:

rmp→ priority -= 1; /\* lower priority \*/

Now in order to balance the increase priority, we need to make sure the queue length is not overflowed. For that, we can check the balance queues function and change the following lines:

rmp→priority -= 1; /\* increase priority \*/

Here, we need to comment the decrement here or else the priority queue will be overflowed.

1. Workload\_mix1.sh

!/bin/sh

./arithoh.sh &

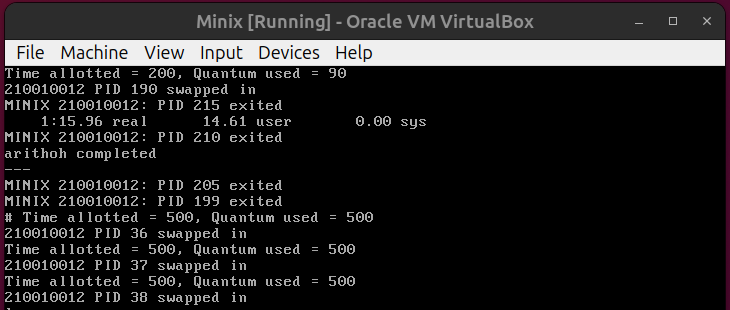
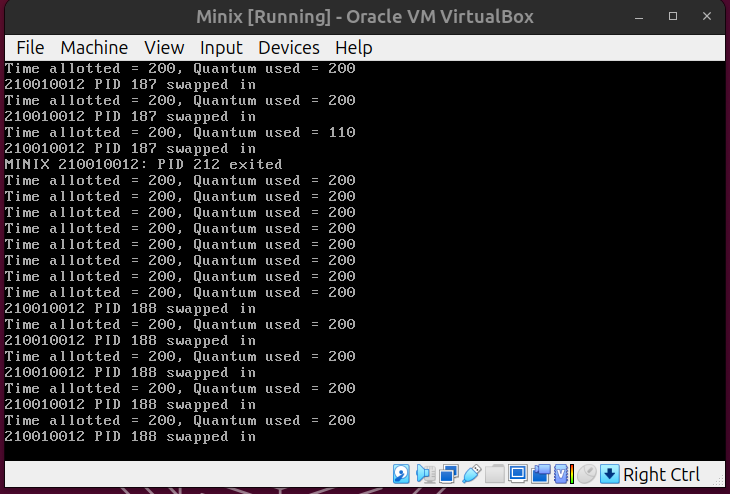
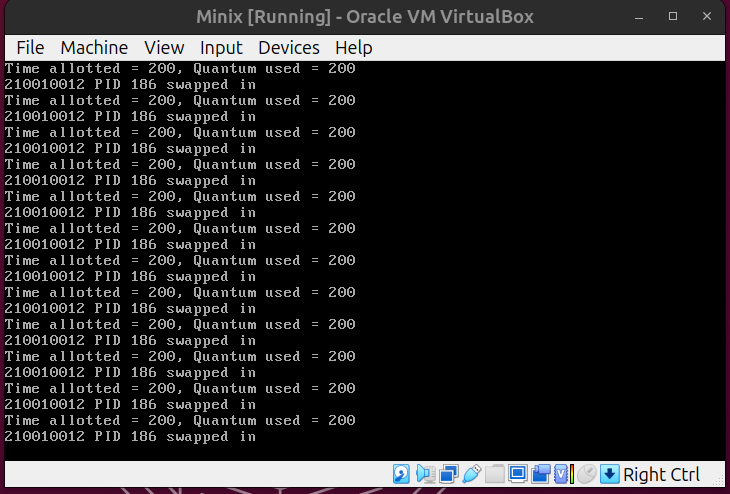
./arithoh.sh &

./arithoh.sh &

./arithoh.sh &

./arithoh.sh &

wait



All five processes are CPU-intensive and execute sequentially in a first-come-first-serve fashion. They fully utilize their time slice of 200, as there are no I/O waits.

2. Workload\_mix2.sh

!/bin/sh

./arithoh.sh &

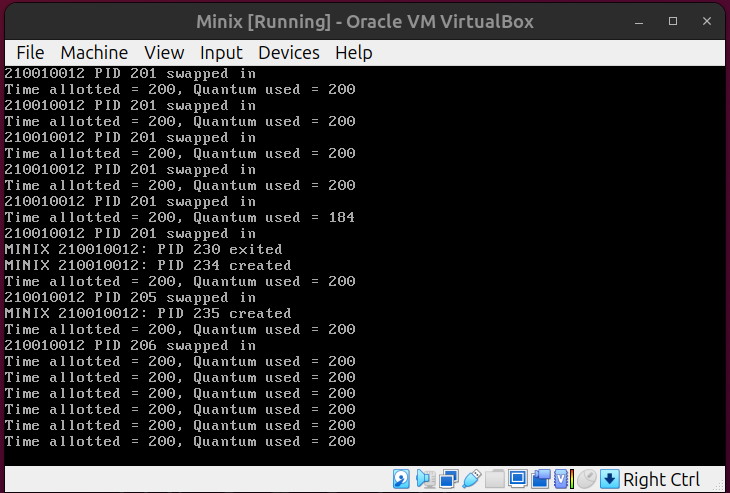
./arithoh.sh &

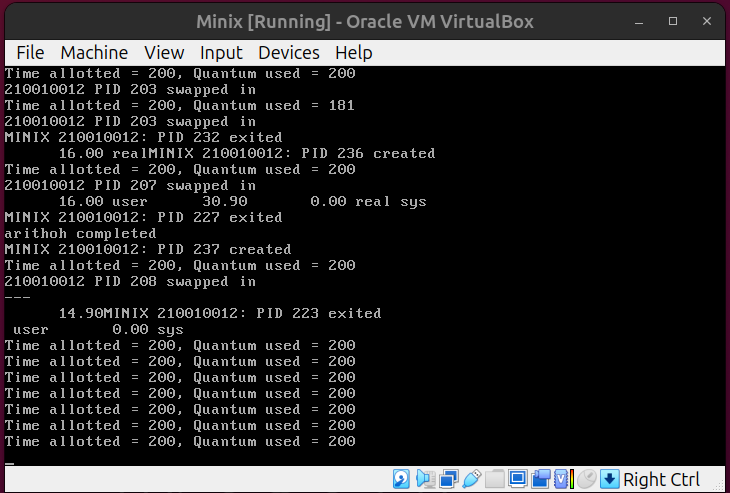
./arithoh.sh &

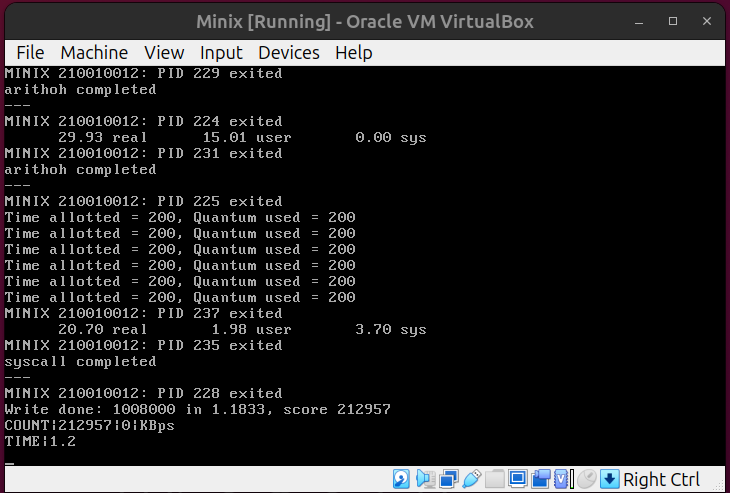
./fstime.sh &

./syscall.sh &

wait







In this scenario, "arithoh.sh" is repeatedly scheduled until its execution is complete, with consecutive scheduling. "syscall.sh" finishes before "fstime.sh" because I/O-bound processes are placed in the waiting queue after requesting I/O, then returned to the ready queue and scheduled for CPU execution. Finally, "fstime.sh" completes last after receiving I/O.

3. Workload\_mix3.sh

!/bin/sh

./syscall.sh &

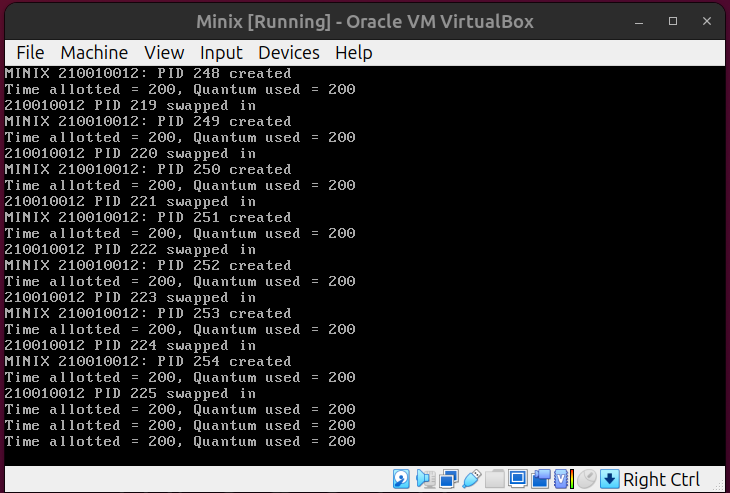
./syscall.sh &

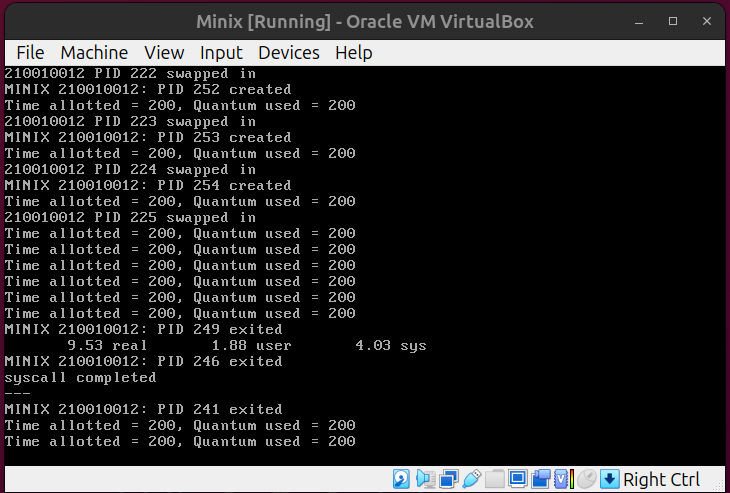
./syscall.sh &

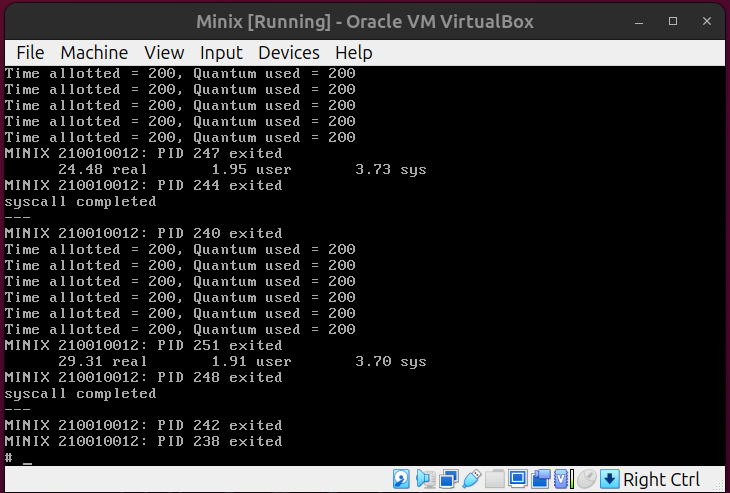
./syscall.sh &

./syscall.sh &

wait







In this scenario, all five processes are CPU-intensive and execute sequentially in a first-come-first-serve fashion. Each process fully utilizes its time quantum of 200, as there are no I/O waits. The next process is scheduled only after the termination of the previous one.

4. Workload\_mix4.sh

!/bin/sh

./fstime.sh &

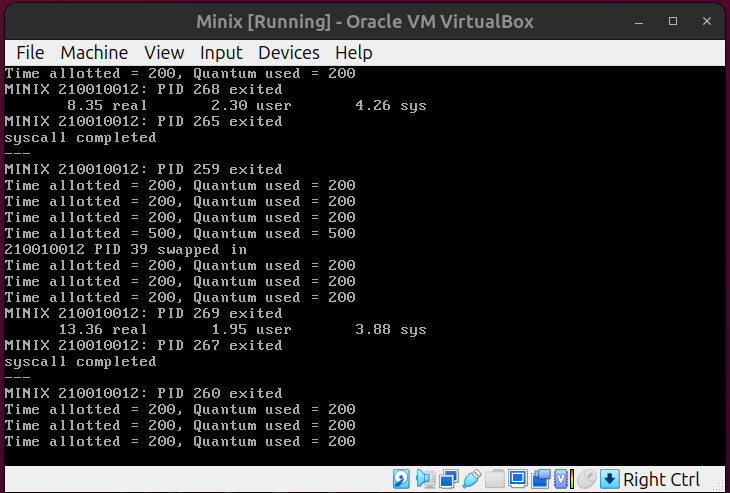
./fstime.sh &

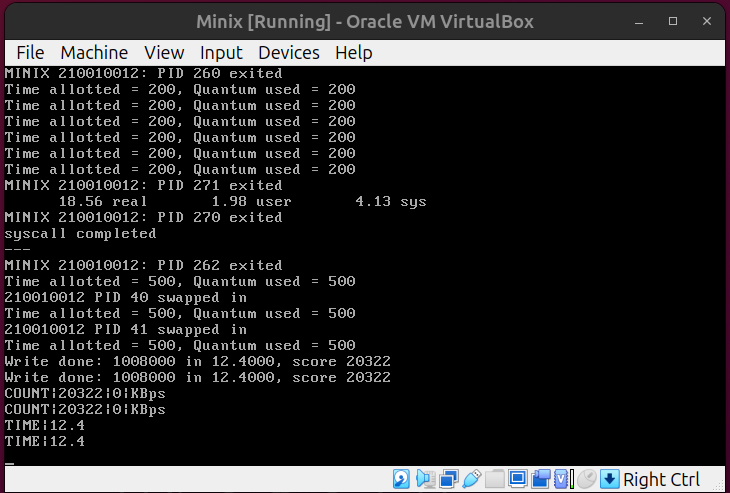
./syscall.sh &

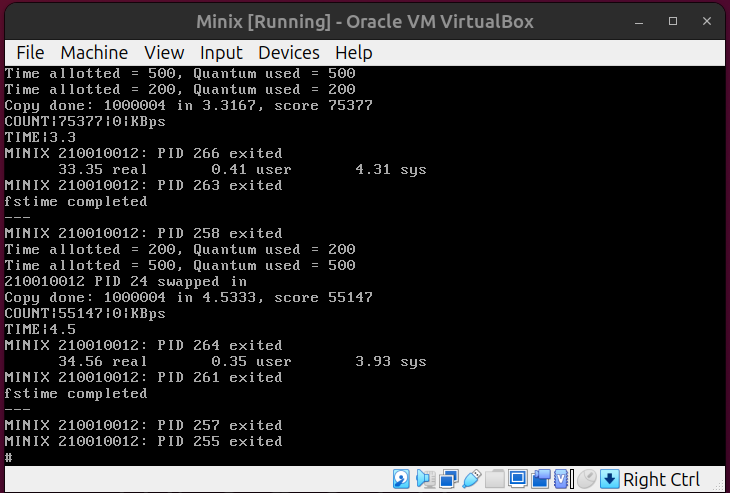
./syscall.sh &

./syscall.sh &

wait







In this scenario, three CPU-intensive "syscall.sh" processes execute before two I/O-intensive "fstime.sh" processes. The I/O-bound processes are placed in the waiting queue after requesting I/O, then returned to the ready queue and scheduled for CPU execution.

5. Workload\_mix5.sh

!/bin/sh

./fstime.sh &

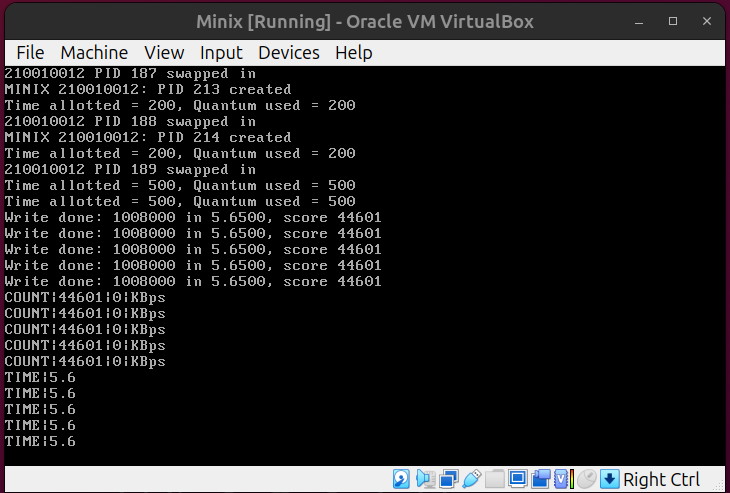
./fstime.sh &

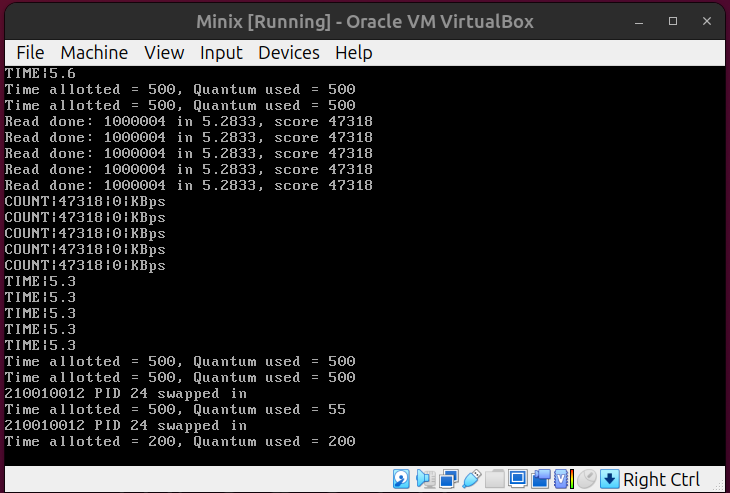
./fstime.sh &

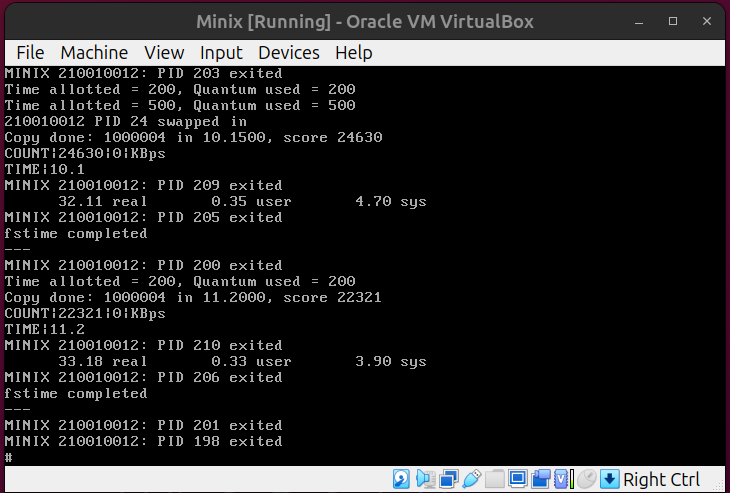
./fstime.sh &

./fstime.sh &

wait







In this scenario, all five processes are I/O-bound "fstime.sh." The output suggests that the Pseudo FIFO order is not strictly followed for I/O-bound processes. Due to this deviation, the process is termed as Approximation to FIFO Implementation or Pseudo FIFO Implementation.