Assignment #3: Part 1 Report

Course Code: SYSC-4001

Lecture Section: A

Lab Section: L3

Group Number: 18

Student 1

Name: Daniel Kuchanski

Student Number: 101182041

Student 2

Name: Jacob Gaumond

Student Number: 101240517

To compare the External Priority, Round Robin, and External Priority with Round Robin scheduling algorithms, 20 test cases were made. These test cases were designed to simulate chosen scenarios shown in the lecture material to observe how the three scheduling algorithms may differ when dealing with them. The test scenarios differed in how they introduced processes of different priority, CPU and I/O bound processes, the order in which processes were introduced, and more.

To begin, a simple case was observed with test cases 1 and 2. Test case 1 simulates a single process with a short CPU burst of 10 ms and no I/O, while test case 2 simulates the same process with a short I/O burst of 1 ms every 5 ms. All three schedulers dealt with these two cases identically. Test case 1 shows a turnaround time of 10ms, as it was scheduled immediately with 0 wait time. Test case 2 shows a similar behavior, except the turnaround time is raised to 11 ms to account for the 1 ms of I/O time.

Test case 3 takes away the I/O once again, but introduces a second process. The first process has a total CPU time of 200 ms. The second process has a higher priority than the first, but arrives 3 ms after. The throughput of all three algorithms is the same, as all three algorithms complete 2 processes in 220 ms. However, their behavior is slightly different. With the External Priority scheduler, preemption is not present. Since all processes are completely CPU bound, the first process executes to completion before the second can begin. However, both of the Round Robin schedulers preempt this process at the first opportunity (i.e., 100 ms). Due to this preemption, the Round Robin schedulers are able to drastically increase the average response time. Specifically, in External Priority the average response time is 100 ms (i.e., (0 ms + 200 ms)/2), while the Round Robin schedulers have half the average response time (i.e., (0 ms + 100 ms)/2). Preemption can clearly drastically increase the response time of high priority processes stuck behind lower priority processes, especially when the lower priority one is CPU bound. Additionally, this can allow the entire system's average response time to decrease.

Test case 5 shows three processes of length 50 ms, 50 ms, and 10 ms, with no I/O interaction. Since no preemption occurs before 100 ms of execution, and no I/O occurs, all three schedulers default to FCFS behavior and perform identically. This shows the importance of choosing a relevant time quantum for Round Robin schedulers in order to allow preemption of processes to produce more diversity in the CPU and a lower average response time for the overall system.

Test case 18 has five identical processes of varying priority, with 100 ms lengths and 20 ms I/O's every 50 ms. Again, all three schedulers behaved identically. Since all of the processes arrived within the first 40 ms, by the time the first process joins the waiting queue at 50 ms all processes are in the system. After that, no process runs long enough to be preempted. I/O performs regularly, but again all scheduling algorithms produce the exact same output. This further proves the point of the importance of shorter time quantums.

Test case 17 attempts to invoke some sort of a convoy effect by having a low priority 500 ms long process enter the system before seven higher priority processes of 30 ms of CPU time. The 500 ms process requires 20 ms of I/O every 100 ms, while the others require 20 ms of I/O

every 50 ms. In all three cases, the high priority process is preempted or sent to I/O at 100 ms, yet Round Robin without priority was the only one to allow a variety of processes to enter the CPU after the high priority process returned. Due to this, Round Robin forces the high priority process to have a much higher waiting time than the other schedulers. In fact, most of its execution occurs after the other seven processes have finished. Due to this, there are no spare processes to take the CPU when the high priority process performs I/O. This causes a significant amount of wasted CPU time. While both the External Priority schedulers end with a throughput of eight processes in 710 ms, the Round Robin scheduler ends with a throughput of eight processes in 770 ms. That shows 60 ms of wasted time for the Round Robin scheduler. This leads one to draw the conclusion that, without priority-based scheduling, Round Robin scheduling can cause significant wait times and lack of resources for important processes, along with overall system inefficiency in some cases. Ironically, although this test case attempted to invoke the convoy effect where a long process blocks many short ones after it, after I/O Round Robin scheduling caused the long process ended up being blocked by the short ones that were originally behind it.

It should be noted that the previous case, like many of the others, would also have been improved by a shorter time quantum. If this were the case, the high priority process would have a chance at the CPU when the 30 ms processes were preempted. However, overall the effect would remain in this case. That being said, this does appear to be a significant trend that is seen across all of the test cases. In many cases the schedulers perform similarly if not identically, although they are built to schedule processes fundamentally differently – if given the chance, that is. Since many regularly-ran computer programs run in a fraction of the time as what was provided for the test cases' total CPU times, one can see that a 100 ms time quantum is completely inappropriate for most computing. Priority scheduling can ease this (such as for for processes which require lower response times), however it is difficult to give priority scheduling the chance without preemption, or with preemption that does not occur in a timely manner.

A further analysis was done specifically on External Priority scheduling. The remainder of the report focuses on this analysis.

For one process with a small CPU burst, the system behaves similarly to the other two, in the test with one process with CPU burst 10 and no I/O, the turnaround time was 10 as it was scheduled immediately and had a 0 wait time. For a process with an I/O burst, the behaviour is similar except the turnaround time increases by 1ms to account for the time spent waiting for I/O (which is 1ms duration).

When we introduce a second process, since we are not using preemption, if a longer process with a higher PID (lower priority) arrives before a higher priority one, it will run its full length if it does not use I/O. However, if the initial process gives up the CPU to wait for I/O, as tested with processes 10, 1, 0, 5, 2, 3 and 1, 2, 3, 5, 0, 0, it will have to wait for the new process to finish entirely, which is where we begin to observe increased waiting time (PID 10 waiting time is 3ms longer than when it had no I/O.

If one varies processes that use high and low memory, waiting time can increase even more since a short but large process will have to wait for higher priority processes on top of waiting for free memory partitions.

For CPU-bound processes, because there is no preemption, a CPU-bound process occupies the CPU until completion, even if higher-priority (lower PID) processes arrive during its execution. So one has high throughput (longer to complete the same amount of jobs) when long CPU-bound processes are running, less response time due to not waiting for I/O, but wait time may increase as mentioned due to memory constriction and the convoy effect.

For I/O-bound processes, there are more transitions, as shown in the output, and lower CPU utilization (processes spend longer in the WAITING queue than running). Turnaround time is better here because other processes can run while the CPU is not being used, the EP scheduler handles I/O-bound processes efficiently only when the CPU is idle. Also, because there is still at least some CPU time, a process can still wait significantly.

The tie-breaker in this case is FCFS, although in this system specifically this is impossible to test due to the fact that priority is based on PIDs, which are unique, so no two processes will ever have the same priority and will therefore not behave in the FCFS fashion.

The EP scheduler can create convoy effects by favouring processes with low PIDs that have long execution times. Even high-priority processes wait for memory or if the CPU is busy, so higher than average wait time.