SYSC4001 Assignment 3 Group 10

Student 1: James Gohl 101299043
Student 2: Magdi Hajjaj 101299841
Date: 01-12-2025

Part 1 – Building a small Scheduler

For part 1 of assignment 3, the objective was to design and implement a small scale operating system scheduler simulator. The goal of this simulator is to be used to analyze the influence of various scheduling algorithms and to gain a better understanding of how scheduling works. For this assignment the simulator was built on the template code in CPP and various tests were simulated.

The CPP simulator works by taking input from a file which contains the details of various processes. These details include its PID, memory size, arrival time, total CPU time, I/O frequency and I/O duration. These values will be used to simulate the various scheduling algorithms for the simulator. The simulator works by first reading the value and passing all of the process information to the simulator function. The simulator will continue to run until all of the process have terminated. When a new process arrives it will assign it memory and it will add it to the ready queue. The simulator will then look through the wait queue to determine if any processes have finished waiting for I/O to be done. If I/O is complete it will then add that process to the ready queue. Next the simulator will act as a scheduler and determine if a process is running and should be kicked out, should be kept running, or bring in another process if none are currently being run. This part of the simulator will also determine if a process is to go into the waiting queue based on its I/O frequency. Three scheduling algorithms were implemented; round robin, external priorities without preemption and a combination of eternal priorities and time in round robin fashion. Throughout the simulator, the code will keep track of each time a process changes state and will output those changes into a file.

After the simulation was created, various test cases were performed, and the results were analyzed to compare the differences between the various scheduling algorithms.

Test 1: Test 1 was a simple simulation with just 1 process that did not include I/0. Since there was only one process for this simulation, each scheduling algorithm resulted in the exact same output. However if the process would have had cpu time that was greater than 100, the output would have slightly varied. This would be due to the process of the timer interrupt interrupting and calling the scheduler once the quantum time of 100 had passed. However unlike in a real operating system, this simulator did not account for the time it takes for overhead task such as calling the scheduler, so the resulting time would still end up being the same just with added lines in the output table.

The calculations for all three scheduling algorithms are as follows:
Throughput: 1 process per 10 ms.
Average Wait Time: 0ms
Average Turnaround Time: 10ms
Average Response Time (time between 2 I/O operations): there is no I/O for this test case:

Test 2: Test 2 was a simple simulation with just 1 process that included I/0. For the same reasons as test 1, the output for test 2 was the same for each scheduling algorithm

The calculations for all three scheduling algorithms are as follows:
Throughput: 1 process per 11ms.
Average Wait Time: 0ms
Average Turnaround Time: 11ms
Average Response Time (time between 2 I/O operations): there was only 1 I/O operation:

Test 3: Test 3 was again a simple simulation with only 2 processes that did not include I/O. For this test the round robin and the external priority algorithm both result in the same output due to the fact that the processes CPU time was not long enough to reach the kick out time of 100ms and there was no prevention for the process with the higher priority. However, the external priority with the round robin algorithm was different as the second process to enter the ready queue had a higher priority and thus kicked out the first one.

	RR and EP	EP with RR
Throughput	2 processes / 15ms	2 processes / 15ms
Average Wait Time	(7 + 0) / 2 = 3.5ms	(0 + 5) / 2 = 2.5ms
Average Turnaround Time	(10+(15-3))/2 = 11ms	(15 + (8-3)) / 2 = 10ms
Average Response Time	no I/O	no I/O

Test 4: Tests 4 was similar to test 3,but this time one of the processes included I/O. Each algorithm for this test resulted in the same output. The round robin and the the eternal priority algorithm were due to the same reasons as in test 3. For this test, the external priority with round robin was the same as the other algorithms due to the fact that the higher priority task did not have the opportunity to kick out the lower priority task as it was already in the I/O wait queue by the time it arrived.

The calculations for all three scheduling algorithms are as follows:
Throughput: 2 processes per 14ms = 1 process / 7ms.
Average Wait Time: (0 + 3) /2 = 1.5ms
Average Turnaround Time: (14 + (8 - 3))/2 = 9.5 ms
Average Response Time (time between 2 I/O operations): for process PID 10: 13 - 2 = 11. There was 11 ms between when the first I/O started and the second one finished.

I/O Bound Tests: Various tests were performed for processes that were I/O bound and the results were examined. The results showed fairly similar results for the three algorithms. This makes sense as for I/O bound processes they spend most of their time performing I/O and thus the CPU is free most of the time. However, this can vary if there is an I/O bound task paired with CPU bound tasks, but if all the tasks are I/O bound then they have similar results.

Large CPU Burst Test: Since the previous tests all had processes that had cpu time of less than 100ms, the round robin scheduler never had the opportunity to kick out a process from the ready queue. Thus test were run to demonstrate what happens when the time limit is reached and a process is kicked out of the ready queue. Tests were performed with large CPU bursts that included I/O and with ones that did not.

Same Priority Test: An attempt was made to make a test case that included two processes with the same priority to demonstrate how that works in the algorithms that deal with priorities. However, it was discovered that due to the nature of how the simulator was built and based on the template code, this was not possible. This is because the instructions were to use the PID as the priority and the code does not allow there to have duplicate PIDs without there being issues when syncing the ques and various other tasks. The possibility of modifying the simulator to allow for duplicate priorities was examined, but due to time constraints this feature was not implemented.

Other Tests: Various other tests were also performed and the results were analyzed. Examples of these tests include processes with similar I/O and CPU times, and various other random test cases.

Conclusion: In conclusion this simulator helped aid in practicing programming and the understanding of operating system scheduling algorithms and how they work. The most useful part of the simulator was building it. A good understanding of the scheduling algorithms was already understood from the lectures, but it is always a good idea to practice programming skills even if the simulator only helped a small amount with understanding the course content. It is also worth noting that the test case did not go super in-depth, but the basic understanding of the algorithms can be extracted from the results. For example, this simulator can show how without preemption if there is a process with a long CPU usage time a process of higher priority can get stuck waiting for it to finish leading to undesirable results. Overall, It was a challenge to design the simulator, but the struggle helped with many important programming skills while confirming understanding of the scheduling algorithms discussed in class.

Github Repository links:

SYSC4001_A3_P1: https://github.com/JamesGohl/SYSC4001_A3_P1