The main goal of this part of the assignment was to design a scheduler simulator that behaves like a simplified operating system. The simulator manages processes, memory partitions, CPU bursts, and I/O events. To make the testing meaningful, we implemented three different scheduling algorithms and ran a total of twenty test scenarios. These scenarios were chosen to cover a wide range of system behaviours, including CPU-bound workloads, I/O-bound workloads, mixed job types, and cases with both heavy and light memory pressure. Throughout the testing, we recorded several metrics such as waiting time, turnaround time, response time, throughput, and overall memory usage. By comparing these values, we were able to understand how each algorithm schedules processes and how the fixed-partition memory model affects the whole system.

The memory system uses six fixed partitions ranging from 40MB to 2MB. Because the partitions cannot be resized, a process can only be loaded into one partition that is at least as large as the memory it needs. This restriction created internal fragmentation quite often. Small processes sometimes ended up using large partitions simply because the appropriate smaller ones were unavailable at the time. This also meant that larger processes occasionally had to wait a significant amount of time for one of the big partitions to become free. Even when the scheduler itself was fair, delays still happened because a process simply had nowhere to load. This interaction between scheduling and memory availability ended up being a bigger factor than originally expected. In some workloads, especially when many processes arrived close together, memory availability had just as much impact as the scheduling algorithm itself.

We tested three scheduling algorithms, External Priority, Round Robin, and a hybrid EP+RR method. External Priority treats lower PID values as a higher priority. EP is non-preemptive, meaning that once a process gets the CPU, it will keep running until its CPU burst finishes or it goes to I/O. It does not voluntarily give up the CPU for another process. Round Robin works very differently. It uses a fixed 100ms time quantum and cycles through all ready processes in order, so each one gets a fair turn on the CPU. RR is preemptive and interrupts processes regularly, preventing any single process from completely taking over the CPU. The hybrid EP+RR scheduler mixes the two ideas. It still respects priority the same way EP does, so the highest-priority ready process is always chosen first. However, if several processes share the same priority, they are scheduled using Round Robin. Unlike EP, the hybrid version is preemptive, and a higher-priority job can interrupt a lower-priority one immediately. This combination allows priority to matter without completely sacrificing fairness.

In our simulations, the EP scheduler behaved exactly as expected for a strict priority system. High-priority processes (low PID values) were almost always the first to run, and they tended to finish quickly because they rarely had to wait behind anyone else. Their response times were excellent since many of them started running almost immediately after arriving. However, the downside was that lower-priority processes sometimes waited a very long time. If a low-PID process had a long CPU burst, it could dominate the CPU and delay every single process behind it. This effect was especially noticeable in CPU-bound scenarios where processes did not block for I/O very often. In those situations, some jobs with higher PID values

waited so long that it noticeably increased the average waiting and turnaround times. EP performed somewhat better in I/O-heavy scenarios because processes naturally released the CPU more often, but the fairness issues never completely went away.

Round Robin produced much more balanced results across the different scenarios. Because each process is only delayed by one 100ms quantum before it gets CPU time, response times were consistently solid. RR prevented starvation entirely, and short jobs tended to finish fairly quickly because they were never stuck waiting behind a long job for too long. RR worked particularly well in workloads where processes frequently switched between CPU and I/O, since the rotation allowed everything to move forward smoothly. However, for CPU-bound workloads with long bursts, RR was noticeably slower. Big jobs had their total execution time stretched out because they were constantly being interrupted. This didn't hurt fairness, but it did increase turnaround times and sometimes lowered throughput compared to EP+RR. Even though RR is simple and predictable, it doesn't adapt well to situations where some processes genuinely need more continuous CPU time.

EP+RR ended up giving the most balanced and consistent performance. The priority system ensured that more "important" processes (in terms of low PID values) still ran ahead of others, but the Round Robin aspect prevented starvation and allowed lower-priority processes to make steady progress. High-priority jobs still benefited from quick response times, but the difference was that lower-priority jobs were no longer delayed indefinitely behind a single long-running process. The hybrid approach also interacted better with memory partition availability. High-priority short jobs finished early and freed their memory partitions sooner, which made space available for new processes that were waiting. This improved memory turnover, especially in mixed workloads where many processes arrived at different times.

Comparing the three schedulers across the twenty scenarios made the differences very clear. In CPU-bound tests, EP created the longest delays for low-priority processes and had the worst fairness. RR handled CPU-bound jobs more evenly but stretched out the completion time for long processes because of constant preemptions. EP+RR avoided both problems, providing reasonable fairness without losing the benefits of respecting priority. In I/O-bound scenarios, all three algorithms performed better overall because the constant I/O naturally gave other processes chances to run. RR and EP+RR were closer in performance in these cases, although EP+RR still generally had slightly better waiting and turnaround times. In mixed workloads, EP+RR clearly stood out. It consistently produced the best combination of quick responses for important tasks and reasonable scheduling for everything else.

Memory behaviour turned out to be an important factor in the results. Internal fragmentation happened often because of the fixed partition sizes. Under EP, a large high-priority process could occupy a large partition for most of the simulation, blocking other large processes and increasing waiting times. RR and EP+RR both tended to free partitions sooner because their scheduling styles allowed more processes to finish earlier. EP+RR had the best memory turnover overall, since high-priority short jobs tended to exit earlier and release the

partitions quickly. This meant that fewer processes were stuck waiting for a usable partition, which helped improve throughput and reduce delays.

In conclusion, after running all twenty scenarios the hybrid EP+RR scheduler delivered the best overall performance. External Priority was very fast for high-priority tasks but suffered from fairness issues, and Round Robin was very fair but slowed down long-running processes. EP+RR successfully combined the strengths of both methods while avoiding most of their weaknesses. It produced strong results across CPU-bound, I/O-bound, and mixed workloads, and it interacted with memory usage in a way that reduced delays and improved throughput. Based on our experiments, EP+RR behaved the closest to what we would expect from a practical, real-world scheduler, making it the most effective and well-rounded option among the three.

Metrics RR:

Scenario	Avg Waiting	Avg Turnaround	Avg Response	Throughput
execution1.txt	0.00	330.00	0.00	0.00303030
execution2.txt	0.00	128.00	0.00	0.00781250
execution3.txt	0.00	1300.00	0.00	0.00076923
execution4.txt	0.00	353.33	0.00	0.00209790
execution5.txt	270.00	565.00	50.00	0.00338983
execution6.txt	77.50	128.75	77.50	0.01951220
execution7.txt	450.00	1425.00	50.00	0.00102564
execution8.txt	0.00	353.33	0.00	0.00209790
execution9.txt	0.00	1300.00	0.00	0.00076923
execution10.txt	2066.67	3166.67	126.67	0.00090909

execution11.txt	0.00	290.00	0.00	0.00344828
execution12.txt	293.33	493.33	100.00	0.00404858
execution13.txt	890.00	1805.00	50.00	0.00055310
execution14.txt	0.00	128.00	0.00	0.00781250
execution15.txt	0.00	330.00	0.00	0.00303030
execution16.txt	270.00	565.00	50.00	0.00338983
execution17.txt	0.00	353.33	0.00	0.00209790
execution18.txt	293.33	493.33	100.00	0.00404858
execution19.txt	450.00	1425.00	50.00	0.00102564
execution20.txt	2066.67	3166.67	126.67	0.00090909

Metrics EP:

Scenario	Avg Waiting	Avg Turnaround	Avg Response	Throughput
execution1.txt	0.00	330.00	0.00	0.00303030
execution2.txt	0.00	128.00	0.00	0.00781250
execution3.txt	0.00	1300.00	0.00	0.00076923
execution4.txt	0.00	353.33	0.00	0.00209790
execution5.txt	270.00	565.00	270.00	0.00338983
execution6.txt	0.00	128.75	0.00	0.01951220
execution7.txt	450.00	1425.00	450.00	0.00102564
execution8.txt	0.00	353.33	0.00	0.00209790
execution9.txt	0.00	1300.00	0.00	0.00076923
execution10.txt	610.00	1800.00	610.00	0.00111111

execution11.txt	0.00	330.00	0.00	0.00303030
execution12.txt	273.33	546.67	273.33	0.00366133
execution13.txt	565.00	1720.00	565.00	0.00058140
execution14.txt	0.00	128.00	0.00	0.00781250
execution15.txt	0.00	330.00	0.00	0.00303030
execution16.txt	273.33	546.67	273.33	0.00366133
execution17.txt	0.00	353.33	0.00	0.00209790
execution18.txt	273.33	546.67	273.33	0.00366133
execution19.txt	450.00	1425.00	450.00	0.00102564
execution20.txt	610.00	1800.00	610.00	0.00111111

Metrics EP_RR:

Scenario	Avg Waiting	Avg Turnaround	Avg Response	Throughput
execution1.txt	0.00	330.00	0.00	0.00303030
execution2.txt	0.00	128.00	0.00	0.00781250
execution3.txt	0.00	1300.00	0.00	0.00076923
execution4.txt	0.00	353.33	0.00	0.00209790
execution5.txt	270.00	565.00	270.00	0.00338983
execution6.txt	0.00	128.75	0.00	0.01951220
execution7.txt	450.00	1425.00	450.00	0.00102564
execution8.txt	0.00	353.33	0.00	0.00209790
execution9.txt	0.00	1300.00	0.00	0.00076923

execution10.txt	610.00	1800.00	610.00	0.00111111
execution11.txt	0.00	290.00	0.00	0.00344828
execution12.txt	206.67	406.67	206.67	0.00491803
execution13.txt	445.00	1360.00	445.00	0.00073529
execution14.txt	0.00	128.00	0.00	0.00781250
execution15.txt	0.00	330.00	0.00	0.00303030
execution16.txt	206.67	406.67	206.67	0.00491803
execution17.txt	0.00	353.33	0.00	0.00209790
execution18.txt	206.67	406.67	206.67	0.00491803
execution19.txt	450.00	1425.00	450.00	0.00102564
execution20.txt	610.00	1800.00	610.00	0.00111111