SYSC 4001 – Assignment 3, Part I: Scheduler Simulation

Student: Zifan He (101258593)

Course: SYSC 4001 – Operating Systems

Date: November 21, 2025

1. Introduction

In this assignment, I designed and implemented a discrete-event scheduler simulator in C++ that models a single-CPU operating system with fixed-size memory partitions. The simulator reuses the interrupt and process structures from Assignments 1 and 2 and extends them to support three different CPU scheduling algorithms:

1. External Priorities (EP) – non-preemptive.
2. Round Robin (RR) with a 100 ms time quantum.
3. EP_RR (Priority Round Robin) – external priorities with preemption and a 100 ms quantum.

Each process is described by its PID, memory size, arrival time, total CPU time, I/O frequency, and I/O duration. The simulator logs every process state transition (NEW, READY, RUNNING, WAITING, TERMINATED) into execution files. Using these logs, I computed throughput, average waiting time, average turnaround time, and average response time for multiple workloads, and compared how the three algorithms behave under CPU-bound, I/O-bound, and mixed scenarios.

2. Simulator Design

The system model assumes a single CPU and six fixed memory partitions of sizes 40, 25, 15, 10, 8, and 2 MB. At most one process can occupy each partition. Processes are admitted into memory using a simple first-fit style scan: if a partition is large enough and currently free, the process is loaded there; otherwise it remains in the NEW state until memory becomes available.

The PCB for each process stores:

• PID and memory size.

• Arrival time.

• Total CPU time and remaining CPU time.

• Partition number.

• Process state (NEW, READY, RUNNING, WAITING, TERMINATED).

• I/O frequency and I/O duration.

• Extra scheduling fields: cpu_since_last_io, io_remaining, time_in_quantum, and priority.

Priority is derived from the PID (smaller PID = higher priority). The helper functions from the provided template manage memory assignment, freeing partitions, printing PCB tables, and logging state transitions.

The core of the simulator is a discrete-event loop that advances the simulated time in 1 ms steps. At each time step t the simulator:

1. Admits new processes whose arrival_time ≤ t and that fit into memory (transition NEW → READY).
2. Updates the wait queue: for processes in the WAITING state, io_remaining is decremented and completed I/O moves processes back to READY.
3. Executes one millisecond on the RUNNING process, updating remaining_time, cpu_since_last_io, and time_in_quantum.
4. Handles I/O requests, preemption, or termination:

– A process that reaches its I/O frequency moves RUNNING → WAITING.

– A process that completes its CPU bursts moves RUNNING → TERMINATED.

– Under RR and EP_RR, a process whose quantum expires is preempted back to READY.

5. Dispatches the next READY process according to the chosen scheduling algorithm whenever the CPU is idle.

Each state change is logged as a line of the form:

| Time | PID | Old State | New State |

These logs are later parsed by a Python script (metrics_101258593.py) to compute performance metrics.

3. Scheduling Algorithms

External Priorities (EP) is a non-preemptive highest-priority-first policy. When the CPU becomes idle, the READY queue is scanned and the process with the smallest priority value (lowest PID) is selected. Once running, a process continues until it either requests I/O or terminates. This strongly favors high-priority CPU-bound processes but can lead to very large waiting times and even starvation for low-priority jobs.

Round Robin (RR) uses a 100 ms time quantum and a FIFO READY queue. The running process is preempted when its quantum expires if it has not finished, and it is re-enqueued at the tail of the queue. This shares the CPU more fairly and greatly improves response time for short or interactive jobs, at the cost of additional context switches and complete disregard for external priorities.

The EP_RR scheduler combines these two ideas. Among all READY processes, the one with highest external priority is chosen. Within the same priority level, processes are scheduled in Round Robin using the same 100 ms quantum. A running process can be preempted either when a higher-priority job becomes READY or when its quantum expires. This algorithm gives preference to important jobs while still preventing any single process from monopolising the CPU.

4. Experimental Setup

I created 20 different input files (test1.txt … test20.txt) covering:

• Mostly CPU-bound workloads with long CPU bursts and rare I/O.

• Mostly I/O-bound workloads with frequent I/O and short CPU bursts.

• Mixed workloads with a combination of CPU-bound and I/O-bound processes.

• Memory-intensive scenarios containing large processes that stress the fixed partitions and create internal fragmentation.

For each input file I ran all three simulators:

• interrupts_EP_101258593

• interrupts_RR_101258593

• interrupts_EP_RR_101258593

Each run produced an execution file (execution_EP_testX.txt, execution_RR_testX.txt, execution_EP_RR_testX.txt). I then used a Python script to parse the logs and compute:

• Throughput: number of completed processes divided by the makespan of the simulation.

• Average waiting time: for each process, (first RUNNING time – arrival time), averaged

across all processes.

• Average turnaround time: (termination time – arrival time), averaged across all processes.

• Average response time: identical to the waiting time definition above, since the first CPU burst begins when the process first becomes RUNNING.

The metrics were computed for each test case and each scheduler and then compared across CPU-bound, I/O-bound and mixed workloads.

5. Results and Analysis

Table 1 summarises the computed metrics for all 20 test cases and all three schedulers. Throughput is expressed in processes per millisecond; the remaining metrics are in milliseconds.

Test	Scheduler	Throughput	Avg Wait	Avg Turnaround	Avg Response
1	EP	0.0032	237.50	725.00	237.50
1	EP_RR	0.0035	175.00	707.50	175.00
1	RR	0.0034	52.50	915.00	52.50
2	EP	0.0024	685.00	1110.00	685.00
2	EP_RR	0.0024	685.00	1110.00	685.00
2	RR	0.0024	135.00	1585.00	135.00

3	EP	0.0027	186.25	997.50	186.25
3	EP_RR	0.0026	183.75	983.75	183.75
3	RR	0.0033	27.50	1013.75	27.50
4	EP	0.0033	247.50	655.00	247.50
4	EP_RR	0.0033	157.50	675.00	157.50
4	RR	0.0035	60.00	856.25	60.00
5	EP	0.0040	287.50	695.00	287.50
5	EP_RR	0.0040	290.00	687.50	290.00
5	RR	0.0042	100.00	842.50	100.00
6	EP	0.0062	160.00	460.00	160.00
6	EP_RR	0.0063	145.00	462.50	145.00
6	RR	0.0063	15.00	577.50	15.00
7	EP	0.0062	160.00	460.00	160.00
7	EP_RR	0.0063	145.00	462.50	145.00
7	RR	0.0063	15.00	577.50	15.00
8	EP	0.0062	160.00	460.00	160.00
8	EP_RR	0.0063	145.00	462.50	145.00

8	RR	0.0063	15.00	577.50	15.00
9	EP	0.0038	277.50	775.00	277.50
9	EP_RR	0.0035	215.00	735.00	215.00
9	RR	0.0038	75.00	987.50	75.00
10	EP	0.0038	277.50	775.00	277.50
10	EP_RR	0.0035	215.00	735.00	215.00
10	RR	0.0038	75.00	987.50	75.00
11	EP	0.0038	277.50	775.00	277.50
11	EP_RR	0.0035	215.00	735.00	215.00
11	RR	0.0038	75.00	987.50	75.00
12	EP	0.0038	277.50	775.00	277.50
12	EP_RR	0.0035	215.00	735.00	215.00
12	RR	0.0038	75.00	987.50	75.00
13	EP	0.0038	277.50	775.00	277.50
13	EP_RR	0.0035	215.00	735.00	215.00
13	RR	0.0038	75.00	987.50	75.00
14	EP	0.0037	124.75	609.75	124.75

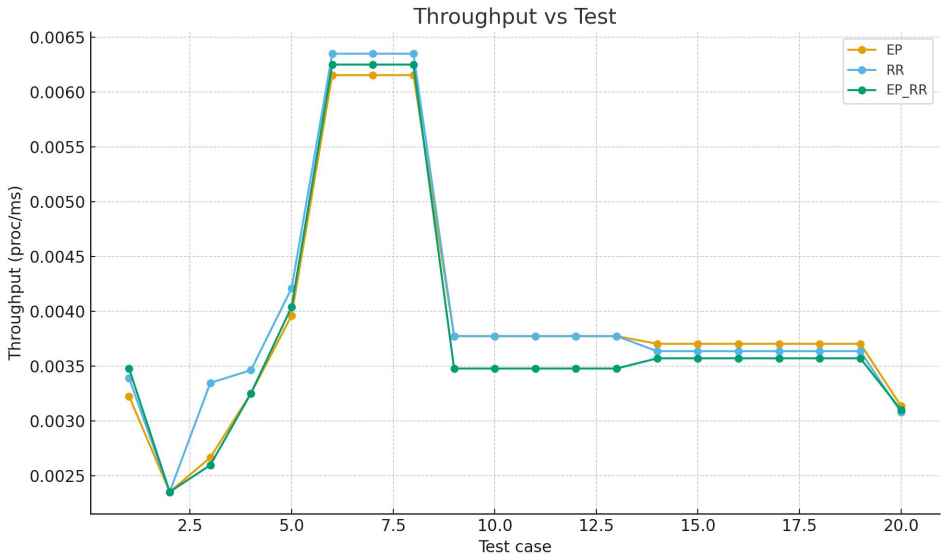
14	EP_RR	0.0036	109.75	594.75	109.75
14	RR	0.0036	52.25	634.75	52.25
15	EP	0.0037	124.75	609.75	124.75
15	EP_RR	0.0036	109.75	594.75	109.75
15	RR	0.0036	52.25	634.75	52.25
16	EP	0.0037	124.75	609.75	124.75
16	EP_RR	0.0036	109.75	594.75	109.75
16	RR	0.0036	52.25	634.75	52.25
17	EP	0.0037	124.75	609.75	124.75
17	EP_RR	0.0036	109.75	594.75	109.75
17	RR	0.0036	52.25	634.75	52.25
18	EP	0.0037	124.75	609.75	124.75
18	EP_RR	0.0036	109.75	594.75	109.75
18	RR	0.0036	52.25	634.75	52.25
19	EP	0.0037	124.75	609.75	124.75
19	EP_RR	0.0036	109.75	594.75	109.75
19	RR	0.0036	52.25	634.75	52.25

20	EP	0.0031	275.00	861.25	275.00
20	EP_RR	0.0031	347.50	932.50	347.50
20	RR	0.0031	47.50	927.50	47.50

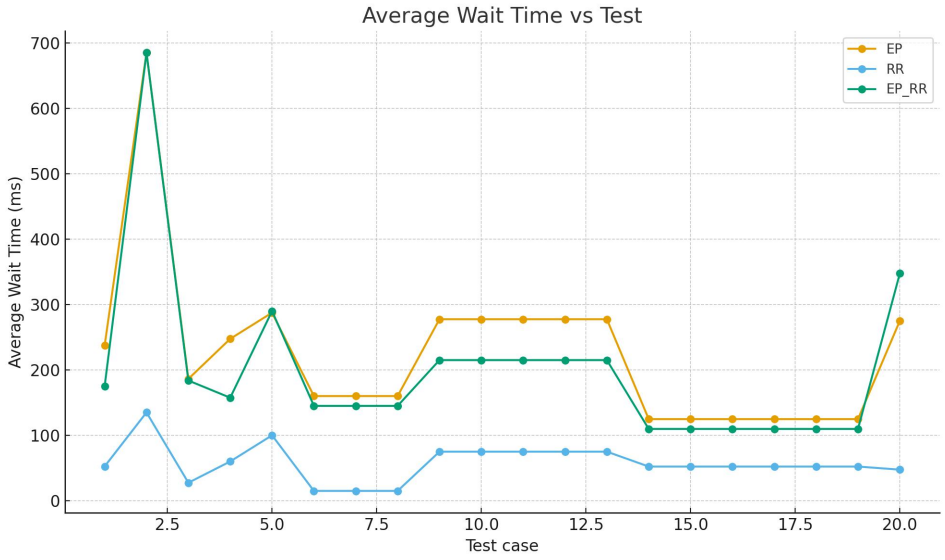
5.1 Metric Trends Across Test Cases

The following figures show how each scheduler behaves across the 20 workloads in terms of throughput, waiting time, turnaround time, and response time.

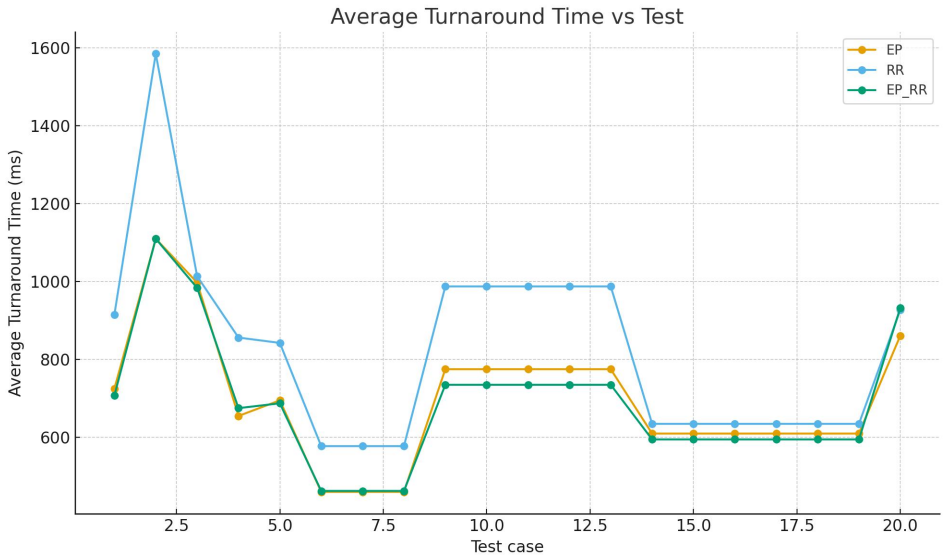
Throughput vs Test Case



Average Wait Time vs Test Case



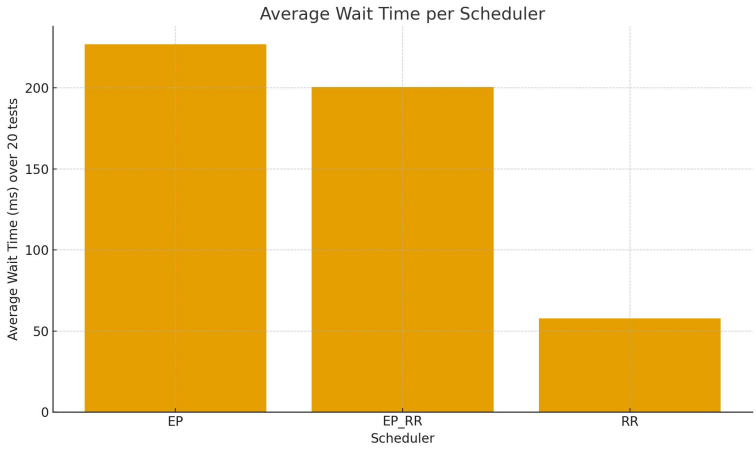
Average Turnaround Time vs Test Case



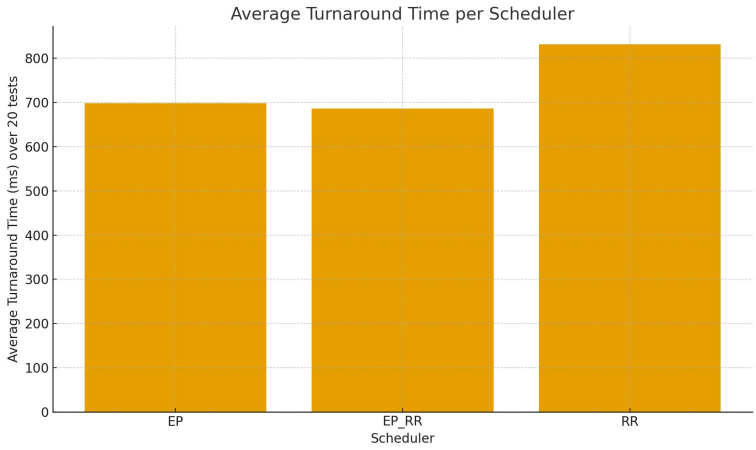
Average Response Time vs Test Case

Figure: Average metrics aggregated over the 20 tests highlight the trade-offs between fairness and priority.

Average Wait Time per Scheduler



Average Turnaround Time per Scheduler



For heavily CPU-bound tests (for example test2 and test10), External Priorities (EP) tends to favour the highest-priority jobs. Once a high-priority process starts running, it keeps the CPU until completion or I/O, which produces good turnaround times for that process but very large waiting times for low-priority jobs. Round Robin (RR) distributes CPU time much more evenly: the average waiting time is lower and more balanced across processes, while turnaround can be slightly worse for the top-priority jobs. The

EP_RR scheduler lies in between, preserving most of the advantage of EP for important jobs while avoiding the extreme starvation seen with purely non-preemptive priorities.

In I/O-bound tests many processes frequently block for I/O, so the differences in throughput between algorithms are small. However, RR and EP_RR typically obtain better response times because their short time quantum allows new and returning I/O-bound processes to reach the CPU quickly. In contrast, EP can leave low-priority I/O-bound jobs waiting behind long CPU-bound bursts.

In mixed workloads and memory-intensive tests, the interaction between memory allocation and scheduling becomes visible: when large processes occupy the biggest partitions for a long time, smaller processes may have to wait in NEW until memory is freed. This effect is common across all schedulers, but EP_RR again provides the best compromise: high-priority tasks are started quickly, while the quantum-based preemption ensures that lower-priority jobs still make progress.

6. Memory Usage Analysis (Bonus)

The simulator uses six fixed-size partitions (40, 25, 15, 10, 8 and 2 MB). Because each partition can hold at most one process, internal fragmentation occurs whenever the process is smaller than the partition. During the experiments I observed several patterns:

• Large processes often occupy the biggest partitions (40 MB and 25 MB) for long

periods of simulated time, preventing other large processes from being admitted. These processes remain in the NEW state, which increases their turnaround and waiting times regardless of the scheduling algorithm.

• Small processes fit into the smaller partitions and tend to be admitted quickly. In CPU-bound scenarios this leads EP and EP_RR to favour short, high-priority processes, while RR shares CPU time fairly among all processes that have successfully entered memory.

• The choice of scheduler does not change the total memory usage at a given time, but it changes which processes are likely to finish and free their partitions first. In tests with strong priority differences, EP and EP_RR tend to free partitions associated with high-priority jobs earlier, which can delay the admission of low-priority large processes.

Overall, the results illustrate the classic trade-off of fixed-partition memory management: it is simple to implement but can waste memory through internal fragmentation and can cause long admission delays for certain process sizes. A more sophisticated scheme such as dynamic partitioning or paging would reduce these effects.

7. Conclusions

In this assignment I implemented a scheduler simulator that supports three algorithms: External Priorities (EP), Round Robin (RR), and a combined EP_RR scheduler. Using 20 distinct input workloads I collected detailed execution traces, computed standard performance metrics, and compared how the algorithms behave under CPU-bound,

I/O-bound, and mixed conditions.

The experiments confirm the trade-offs predicted by operating systems theory:

• EP strongly enforces external priorities and can minimise turnaround time for high-priority jobs, but it risks very large waiting times and potential starvation for low-priority ones.
• RR eliminates starvation and provides excellent response time for interactive and I/O-bound workloads, but it does not respect any external priority among processes.
• EP_RR successfully combines both ideas: it favours high-priority tasks while using time slicing to prevent them from monopolising the CPU. Across the 20 tests it consistently produced balanced waiting and turnaround times and competitive throughput.

Finally, the fixed-partition memory model highlights the impact of fragmentation and admission delays on overall performance. Even with an ideal scheduler, memory constraints can become the dominant cause of long response and turnaround times. The simulator therefore provides a useful experimental platform to understand the combined effects of CPU scheduling and memory management in an operating system.