SYSC 4001: Assignment 3, Part I
Scheduler Simulation Report
Student ID: 101268686, 101311227
Course: SYSC 4001: Operating Systems, Fall 2025

1. Introduction
The purpose of Part 1 of the assignment was to build a small scheduler simulator. The system has a single CPU and a user space of 100 MB divided into 6 sizes: 40, 25, 15, 10, 8, and 2 MB.
We compared with three schedulers:

1. External Priorities (EP): non-preemptive, lower PID = higher priority.
2. Round Robin (RR): time-sharing with a fixed time quantum.
3. External Priorities + Round Robin (EP_RR): priority-based scheduling with preemption and a time quantum.

For each scheduler, we simulated 20 different traces and recorded the process state transitions in execution logs. From those logs we computed the following metrics:

- Throughput, Average Waiting Time, Average Turnaround Time, Average Response Time

2. Simulator Design: 2.1 Data Structures
We made fixed partitions in a table of structs: partition_number (1–6), size (40, 25, 15, 10, 8, 2 MB), code (string: "free", "init", or the program name / PID)
We also added a hook to log memory usage when a process transitions from NEW → READY.

Each process is represented by a PCB that stores the fields for this assignment:

- PID, arrival_time, processing_time (total CPU time), remaining_time (for preemption), io_freq and io_duration, state (NEW, READY, RUNNING, WAITING, TERMINATED), start_time and finish_time (for response and turnaround), partition_id

Queues are represented by std::vector<PCB>: **ready_queue, wait_queue** (for I/O), **job_list**

2.2 Process State Simulation
The main loop moves a **discrete time** variable also known as current_time which starts from 0 ms and att each step, the simulator:

- **Brings new arrivals** in (arrival_time == current_time)
    - Calls assign_memory(p), Changes state NEW → READY, pushes process into ready_queue and job_list, logs the transition in the execution file, logs memory usage in memory_log.txt
- If I/O completion time = current_time then WAITING → READY.
- **Selects the next process** if the CPU is idle.
- Reduces remaining_time, checks if it should go to I/O (RUNNING → WAITING), checks if it has finished (RUNNING → TERMINATED) and for RR and EP_RR, checks if the time quantum has expired (RUNNING → READY).

Each state change is recorded in an **execution log**: | time | pid | old_state | new_state | and these logs are used for calculations.

3. Scheduling Algorithms
3.1 External Priority (EP): EP is a **non-preemptive priority scheduler**. When the CPU is idle, the algorithm picks the READY process with the **lowest PID** and runs it until:

- It ends or It does an I/O and moves to WAITING.

No other process can preempt a running process even if it has a lower PID.
3.2 Round Robin (RR): RR uses a **fixed time quantum, and t**he ready queue is like a FIFO:

- When a process is chosen, it goes for one time quantum but if it does **not** complete within the quantum and does **not** request I/O, it is preempted and is sent to the **back** of the queue but if it does I/O or finishes, it leaves the CPU.

3.3 External Priority + Round Robin (EP_RR)

- The **highest priority** (lowest PID) is chosen but the process still runs under a **time quantum** and if the quantum expires and the process isn't finished or is still waiting for an I/O then it is preempted and returns to READY.

This is used to keep the **responsiveness** of RR while still giving preference to higher-priority processes.

4. Setup and Metrics
We executed at least **20 simulation scenarios per scheduler**:

- EP_traceX.txt, RR_traceX.txt, and EP_RR_traceX.txt contain the execution logs.
- For each log, we ran analyze_metrics.py to parse the transitions and compute.
- The results for every run are stored in metrics_summary.csv.

4.1 Metric Definitions

1. **Throughput:** Throughput= number of processes completed / total simulated time
2. **Average Waiting Time:** how long it stayed in **READY**and averaged over PIDs.
3. **Average Turnaround Time:** Turnaround=finish−arrival
4. **Average Response Time (I/O):** For I/O, we measured interval from RUNNING → WAITING to WAITING → READY, and then averaged it out

Overall,

- **Throughput** ranged from **0.024–0.064 processes per unit time** for multi-process traces
- **Average waiting times** were around **10.67 ms** or **15.33 ms**, depending on the trace.
- **Average turnaround times** around **35–40 ms** for shorter traces and around **64–69 ms** for longer.
- **Average response times** for I/O were usually either about **2 ms** or **3.45 ms**.

Because the same trace sets were run under each scheduler algorithm, we can easily compare EP, RR, and EP_RR.

5. Results and Discussion

5.1 Throughput Comparison

For **single-process traces** all three schedulers got the **same throughput**:

- Because only one process exists, it runs to completion and has no competition.
- Throughput ≈ 0.1, average wait time ≈ 0, and turnaround time is the total CPU time.

For **multi-process traces**, throughput values go into 2 groups:

- **0.062–0.064** for shorter total run times
- **0.024–0.027** for longer simulations.

Across these EP, RR, and EP_RR all get **similar throughput**, which shows that the choice of scheduler does not change the total number of completed processes per unit time.

5.2 Waiting Time and Turnaround

- A**verage waiting times around 10.67 ms** and **turnaround ≈ 64.67 ms**.
- RR and EP_RR runs have **waiting ≈ 15.33 ms** and **turnaround ≈ 69.33 ms** for the same number of processes.

EP reduces waiting time for at least some processes which lowers average turnaround a little bit. **RR and EP_RR have more context switches** which adds an extra delay before a process goes to the CPU, especially if there are many READY processes going through the quantum. However, the differences aren't **huge** because all three schedulers are in the same general range. **Priority scheduling gives a small advantage in average wait/turnaround**, but is balanced because RR and EP_RR are not horrible.

5.3 Response Time and I/O Behavior

The **average response time** is either **2.00 ms** or **3.45 ms** across all schedulers.

- I/O devices are always available (no queue)
- The main effect of the scheduler is when a process **returns to READY** after I/O and has to wait to be scheduled again.

Round Robin and EP_RR rotate through processes, so they can give **more CPU slices** to I/O-heavy processes, but this did not produce a large difference compared to EP. All three schedulers have low response times.

5.4 CPU-Bound vs I/O-Bound Processes

- **CPU-bound processes** benefit from **priority scheduling** (EP) because once they start running, they aren't preempted and can finish quickly which reduces their turnaround time which is why there were lower waiting times observed for EP traces.
- **I/O-bound processes** benefit from Round Robin behavior because if a process blocks for I/O, RR/EP_RR then it will make sure that other processes also get CPU slices, but when the I/O-bound process becomes READY again, it will be scheduled quickly into the rotation.

EP_RR **preempts long-running processes** using a quantum model that behaves similarly to pure RR which shows that all processes are similar in terms of CPU and I/O bursts.

6. Memory Usage (Bonus)

For the bonus part, we added logging of memory usage whenever a process is admitted (NEW → READY). So that means that the simulator assigns the process to one of the six fixed partitions according to its memory size and it logs a line to memory_log.txt in the format: TIME=<t> USED=<used_memory> FREE=<free_memory> USABLE=<usable_memory>

The output files memory_case_X.txt (one per trace) show the times at which processes start and how the memory layout changes We found that large processes can only fit into the **40 MB** or **25 MB** partitions so if those are occupied then they wait even if smaller partitions are free which leads to **internal fragmentation**. For example, a 20 MB process in the 25 MB partition wastes 5 MB that no other process can use. This means that the number of admitted processes is bounded by the total partition sizes and their fit, which affects **throughput** and **waiting time**.

7. Conclusion

In this assignment, we built a scheduler simulator with:

- A single CPU and a fixed-partition memory model,
- Three schedulers: EP, RR, and EP_RR,
- Detailed logging of process state transitions and basic memory usage.

We ran at least 20 traces for each scheduler and computed **throughput**, **average waiting time**, **turnaround time**, and **I/O response time** using a Python analysis script.

The results show that:

- All three schedulers get a **similar throughput**.
- **External Priorities (EP)** reduces the average waiting and turnaround times for many traces by letting higher-priority processes run to completion without preemption.
- **Round Robin (RR)** and **EP_RR** gives more fairness with time-sharing, but affects waiting time due to the time quantum, but not a lot of performance loss.
- **Response times** for I/O are low and stable across algorithms, because I/O devices are always available and the fixed I/O duration always dominates.

Overall, the experiments prove that non-preemptive priority scheduling is slightly better for throughput and turnaround processes, Round Robin improves responsiveness, and a hybrid EP_RR scheduler balances both behaviors while preserving the process priorities.