SYSC4001 - Operating Systems
Assignment 3 – Concurrency, Shared Memory, Virtual Memory, Files
Student 1: Lina Elayed - 101297993
Student 2: Noor ElShanawany - 101302793

PART I: https://github.com/CodeitLina/-SYSC4001_A3_P1.git

1- Overview and setup:

Using the provided template, we implemented a complete scheduling simulator that supports all state transitions introduced throughout the course:

NEW → READY → RUNNING → WAITING → TERMINATED

Most of our work involved extending and modifying the “manage wait queue” and “scheduler” sections of the template so that the simulator correctly handles: I/O events CPU preemption Round-Robin time slicing External priority enforcement Memory allocation using fixed partitions
We executed twenty simulation cases, covering: CPU-bound processes I/O-bound processes mixed workloads memory-stress scenarios For every scenario, we calculated the required performance metrics: Throughput Average, Waiting Time Average, Turnaround Time, Average Response Time.
These metrics allowed us to compare how each scheduling algorithm behaves under different workload types.

2- Explanation of generated input and output files:

Cases 1 through 4: from testcases.pdf to ensure our outputs are exactly what is expected.
Then input files were generated to test 3 main categories:

a) CPU-Bound Workloads: Cases 5 through 9
b) IO-Bound Workloads: Cases 10 through 14
c) Mixed Workloads Cases 15 through 10 (longer CPU + short IO VS longer IO + short CPU)
d) Memory Stress Workloads: (Case 3,4, and 20)

I also want to mention that I ran the case case for all 3 algorithms (found on git) to clearly see the change in metrics or memory purely based on nothing other than algorithm. While many of the calculations were unique, there are some trends/ insights we can take away from the algorithms.

3- Discussion on each workload:

a) CPU-Bound:
    - EP heavily favours small PID values. High-priority processes complete extremely fast; lower-priority CPU-bound tasks suffer starvation or very long waits.
    - RR is more fair but suffers from high context-switch overhead with long CPU bursts.

- EP+RR performs best overall because: it gives priority to important jobs it still time-slices to prevent starvation turnaround is significantly lower than EP or RR alone

b) IO_bound
   - Under EP, an I/O-bound PID with low priority can be blocked indefinitely behind a CPU-bound task.
   - I/O-bound processes benefit from RR because each returns to the CPU quickly and fairly.
   - EP+RR still performs well because higher-priority interactive tasks preempt CPU-bound tasks. RR is the best algorithm for I/O-heavy workloads.

c) Mixed
   - EP produces inconsistent results depending on PID distribution. A low-priority I/O-bound task may experience significant starvation.
   - RR is predictable, but CPU-bound tasks dominate quantum usage.
   - EP+RR is most balanced: responsive for important tasks fair for long-running jobs stable turnaround and wait times

d) Memory pressure
   - Many processes waited for Partition 1 (40MB) or Partition 2 (25MB). Some READY tasks could not start due to memory unavailability, increasing turnaround times.
   - EP could worsen memory delays because a low-priority long process occupying a large partition delays higher-PID tasks.
   - RR freed memory more predictably due to frequent preemptions and time slices.
   - EP+RR again produced the most balanced memory utilization patterns.

<u>4. Bonus – Print Memory Logging</u>

The print_memory_log records the total memory used list of occupied partitions total free memory usable free memory state each time a process transitions from NEW to READY. The prominent trends we see is that:

- I/O-bound cases kept memory highly available because processes frequently returned to WAITING to READY cycles, freeing large partitions sooner.
- CPU-bound cases locked partitions for long continuous CPU bursts, significantly reducing system flexibility.
- Mixed tests showed realistic fragmentation where 8MB and 2MB partitions often remained unused because most tasks were larger.
- EP+RR improved memory distribution by allowing preemption and fairness while maintaining external priorities. It really helped me see how memory management interacts with CPU scheduling.

<u>5. Conclusions</u>

After running 20 simulation scenarios through all three schedulers, the following conclusions were reached:

- Round Robin (RR)
    - Best for: I/O-bound workloads interactive tasks fairness
    - Worst for: CPU-bound tasks minimizing context switches
- External Priorities (EP)
    - Best for: systems requiring strict priority enforcement real-time-like constraints (highest-priority must run first)
    - Worst for: fairness starvation avoidance mixed workloads
- EP + RR (Preemptive Priority Round Robin)
    - Best overall scheduler across all scenarios.
        - Strengths: fast response for high-priority tasks avoids starvation good balance between throughput and fairness best performance in mixed and memory-stress workloads
        - Weaknesses: slightly more context switching than EP slightly slower response than pure RR for I/O-only workloads

In conclusion, The simulation demonstrates how different scheduling algorithms impact system behavior for the same cases. And the second biggest take away is that the combination of EP+RR algorithm is the best option as it delivers the most stable and efficient performance, making it the most suitable choice for general-purpose operating systems.

PART II:
https://github.com/Noor-e001/SYSC4001_A3P2
reportPartC

PART III: