Report
Part 1

Shahd Elsaman
101268283
Nawal Musameh
101360700

In this assignment, we simulated three CPU scheduling algorithms: EP, EP_RR, and RR. By testing multiple examples and reviewing different execution traces, performance metrics, and memory-usage patterns, we were able to observe how each scheduling strategy responds to a variety of workloads. The goal of this report is to analyze the results of the simulation tests we ran and compare how each algorithm handles CPU and I/O interactions, manages process timing, and uses system memory. Through this comparison, we build an understanding of which algorithm performs best under which circumstances, and why these differences occur.

For the EP_RR simulation test (Test 20), the results clearly showed how combining priority scheduling with time-sliced influences overall system behavior. In this test case, the first process (PID 10) arrives at time 0 and immediately begins execution because EP always selects the highest-priority process available. However, PID 10 experiences several I/O operations that repeatedly interrupt its execution. Each time it performs I/O, it leaves the CPU and moves temporarily to the waiting state. When PID 4 arrives at time 70, it is inserted into the ready queue according to its priority and begins running as soon as PID 10 enters an I/O phase. This back-and-forth between CPU and I/O shows how EP_RR balances priority where the higher-priority processes still matter, but lower-priority ones are not completely ignored. The CPU is shared in a way that respects priority without allowing one process to block all others.

The performance metrics for EP_RR reflect this pattern. The algorithm produced a throughput of 0.0042 processes per millisecond, an average waiting time of 104 milliseconds, and an average turnaround time of 359.5 milliseconds, with a response time of 10 milliseconds. These numbers show how the high turnaround time mainly comes from PID 10, which is long-running and I/O-heavy. The waiting time is neither extremely high nor extremely low, it is somewhere in the middle because RR helps prevent long time of delay and waiting, however, the priority rules still cause some delays for the lower-priority process.

The memory usage for this test also illustrates how the simulator handles process life cycles. Memory starts at 15 KB when PID 10 is loaded. It rises to 23 KB when PID 4 arrives at time 70, since that process also needs memory. When PID 4 eventually terminates at time 314, the memory usage drops back to 15 KB, and then finally to 0 KB after PID 10 finishes at time 473. This consistent pattern of memory increasing when a process enters, and decreasing when it terminates shows that the simulator properly

allocates and frees memory in fixed partitions, confirming that the memory model is functioning exactly as expected.

On the other hand, the RR algorithm produced different results when running Test 20. In Round Robin, processes share the CPU through a fixed quantum time, meaning there is not any process that can occupy the CPU for too long without being switched. As a result, the response time for RR is significantly better than EP_RR. The RR have 0 milliseconds, meaning the first slice begins immediately at time 0. RR also achieved a much lower average waiting time of 25.5 milliseconds and a turnaround time of 202 milliseconds. These values are better than EP_RR because RR ensures continuous rotation between processes, preventing long domination of the CPU by any one process. The throughput of 0.0057 is also the highest among the algorithms we tested, which makes sense because RR keeps the CPU busy and efficient by overlapping computation across multiple processes without long pauses.

The memory usage in the RR test also reinforces the idea of fairness and steady flow. Memory begins at 15 KB with the arrival of the first process and increases slightly to 17 KB when the second one is loaded. After everything finishes, memory returns to 0 KB. RR does not change memory patterns beyond when processes arrive and terminate, which shows that scheduling decisions do not interfere with the memory allocation system. The smooth increase and decrease in memory usage further confirm that the simulator correctly implements fixed memory partitions and handles them consistently across algorithms.

Finally, the EP algorithm behaves differently compared to both RR and EP_RR, especially when there are very few processes. In the EP Test 22 example, only one process (PID 3) was loaded into the system and it required 8 KB of memory, arrived at time 0, and needed only 4 ms of CPU time. Since EP is non-preemptive and priority-based, the scheduler immediately selected this process as soon as it arrived. The process executed for 2 ms, triggered an I/O event, moved to the waiting state, and then returned to the CPU after the I/O completed at time 3. Because no other processes were waiting, it resumed running immediately and terminated at time 5. This behavior highlights the strengths of EP when workloads are small: CPU access is instant, waiting time is minimal, and the only context switches occur due to I/O. However, this same design can lead to problems in larger workloads, because EP does not preempt. If a long, high-priority process starts running, all lower-priority processes must wait until it finishes or performs I/O, which can lead to extremely long delays or even starvation. In the

simple test case, EP works perfectly, but in more complex systems its weaknesses become more noticeable.

Overall, the simulations show that each algorithm has different strengths. RR provides the best fairness and responsive performance but ignores priority. EP is ideal for urgent or high-priority workloads but performs poorly under heavy competition. EP_RR offers a balanced approach that respects priority while still providing fairness, making it a strong middle-ground solution. Memory usage across all tests behaved predictably and aligned with theoretical expectations, confirming that the simulator accurately models fixed partition allocation and deallocation. Together, these results help illustrate how CPU scheduling choices influence system performance in real operating systems.