Introduction

This simulation models process scheduling and memory allocation in a multiprogramming operating system. Three schedulers were implemented and evaluated:

- **EP (Earliest PID)** – always selects the READY process with the smallest PID.
- **RR (Round Robin)** – allocates CPU time using a fixed time quantum among READY processes.
- **EP+RR Hybrid** – selects the first process using EP, then continues scheduling with RR.

Each process may perform CPU bursts, request I/O operations, and terminate when its required processing time completes. The simulation logs every state transition (NEW → READY → RUNNING → WAITING → TERMINATED) in execution.txt. A fixed partition memory model is used for process admission.

Scheduling Model

All three schedulers operate on the same cycle:

1. **Process Admission**
   - A process in the NEW state is admitted to memory if a partition large enough for its memory requirement is free.
   - Once admitted it becomes READY for scheduling.
2. **CPU Dispatch**
   - At each millisecond if the CPU is idle the scheduler selects a READY process based on the algorithm in use.
3. **Execution and I/O Handling**
   - Each RUNNING process consumes 1 ms of CPU time.
   - If its I/O frequency threshold is reached, it issues an I/O request and moves to WAITING.
   - After the I/O duration is over it returns to READY.
4. **Termination**
   - When a process' remaining time reaches zero, it terminates and its memory partition is freed.

Results and Metric-Based Analysis

Throughput

EP:

- Long low-PID processes frequently delay higher-PID ones from completing.
- tests with many I/O-bound processes show lower throughput because EP keeps one process running for long bursts.
- EP tends to have lower throughput in mixed workloads.

RR:

- Consistently higher throughput across CPU-bound and mixed tests. Highest throughput overall
- RR rotates the CPU frequently, so long bursts are broken up preventing bottlenecks.
- I/O-bound processes benefit by returning to the READY queue quickly.

EP+RR:

- After the first dispatch behavior resembles RR.
- EP+RR achieves throughput very close to RR.

Average Waiting Time

EP:

- Highest waiting times for high-PID processes in nearly all scenarios.
- In workloads with one long CPU-bound low-PID job, waiting time spikes.

RR:

- Lowest waiting time overall because CPU access is shared evenly.
- Even high-burst CPU processes receive regular CPU slices.

EP+RR:

- Initial low-PID process benefits from EP selection.
- After that, waiting times closely follow RR.
- Average waiting time is slightly lower than RR for low-PID processes, but overall similar.

Average Turnaround Time

EP:

- Turnaround is excellent for low-PID jobs, poor for others.

- CPU-bound workloads highlight the imbalance.

RR:

- Larger turnaround time than EP for short CPU-bound jobs.
- Much better turnaround for I/O-heavy and mixed workloads.

EP+RR:

- Slightly better turnaround than RR for low-PID CPU-bound jobs.
- Better turnaround than EP for higher-PID processes.

Average Response Time (I/O responsiveness)

EP:

- I/O-bound processes have poor response time if they have high PIDs, because they wait behind long CPU-bound low-PID jobs.
- Response times vary widely based on PID.

RR:

- Most consistent and lowest response time.
- I/O-bound processes enter READY and are serviced quickly due to round-robin rotation.

EP+RR:

- First response slightly favors low-PID processes.
- After initial burst responsiveness matches RR.

Behavior with Different Workload Types

CPU-Bound Processes

CPU-bound jobs benefit from long uninterrupted bursts.

- **EP:** Performs well for low-PID CPU-bound jobs but poorly for others.
- **RR:** CPU-bound jobs take longer because CPU time is sliced, so more context switches.
- **EP+RR:** Better than RR for the first CPU-bound job and similar afterward.

Best: EP for low-PID CPU-bound processes
Worst: RR for CPU-bound processes

I/O-Bound Processes

These processes repeatedly perform short CPU bursts and frequent I/O.

- **EP:** High-PID I/O-bound processes often get delayed.

- **RR:** Best performance, rapid CPU returns improve response time.
- **EP+RR:** Behaves like RR after startup.

Best: RR
Worst: EP

Mixed Workloads

Contains a blend of long CPU tasks + frequent I/O tasks.

- **EP**: unpredictable, often bottlenecked by one long CPU-bound job.
- **RR**: evenly distributes CPU improving flow of both types.
- **EP+RR**: start is EP and long-term becomes RR.

Best: RR
Worst: EP

Bonus: Memory Usage Analysis

Fragmentation

Because partitions are fixed in size, internal fragmentation occurs when:

- A process requiring 12 MB is placed in a 15 MB partition → 3 MB wasted.
- Small processes occupy medium-sized partitions.

This effect appears in multiple tests, especially those with small memory requirements.

Admission Delays

When large processes arrive early:

- They occupy the 40 MB or 25 MB partition for long durations.
- Later processes requiring similar sizes are forced to wait in NEW.
- Memory logs show large free memory sums but no suitable free partition.

Scheduler Influence

- **EP and EP+RR:** Low-PID long processes block memory for longer causing the highest admission delays.
- **RR:** More even CPU distribution means processes reach termination more evenly so partitions free sooner.

Conclusion

In conclusion, across 20 test scenarios we found that:

- RR is the most stable and fair scheduler across all workload types.
- EP is efficient only for a subset of processes and creates bottlenecks for high-PID jobs.
- EP+RR provides a benefit over RR in very specific scenarios but generally has very similar behaviour.
- Memory analysis confirms that scheduling strategy significantly impacts partition turnover and overall system responsiveness.