Algoritmos e Estruturas de Dados I
Bacharelado em Engenharia de Software
ICEV – Instituto de Ensino Superior

Analysis Report: iCEVOS Process Scheduler

Componentes:
Paulo Vinicius Alves Melo

Teresina-PI
09/2025

1**. Design Justification**

Why is a manual linked list efficient for the scheduler?

1. O(1) insertion at tail – perfect for FIFO queues
2. O(1) removal from head – matches the “next to run” semantics
3. No random access needed – we never need the n-th process
4. Dynamic size – no waste of pre-allocated memory
5. Simple code – easy to debug and prove correctness

Using an array or vector would require O(n) shifts or amortized resizing, making linked lists the natural, efficient choice for this problem.

2**. Big-O Complexity Analysis**

Operation	Complexity	Explanation
insert_end(process)	**O(1)**	**Tail pointer update**
remove_start()	**O(1)**	**Head pointer update**
run_cpu_cycles()	**O(1)**	**Constant work per cycle**
unblock()	**O(1)**	**Single remove + append**
load_process()	**O(n)**	**One pass over file**

All scheduler-critical operations are constant time, ensuring scalability even with thousands of processes.

3**. Anti-Starvation Analysis**

Rule: After 5 consecutive HIGH executions, force one MEDIUM (or LOW if MEDIUM is empty).

Fairness guarantee:

- Every 5 cycles lower priorities get a chance
- Bounded waiting time = 5 × shortest HIGH job
- No process waits indefinitely

Risk without the rule:

LOW/MEDIUM processes could starve indefinitely if HIGH queue never empties (e.g., constant arrival of HIGH jobs).

The rule introduces predictable fairness with minimal impact on HIGH throughput.

4**. Disc Blocking Life-Cycle**

1. Arrival – process enters its priority queue

2. First selection – scheduler detects resource_needed == "DISC" and was_blocked == false

3. Block – process is removed from priority queue and appended to blocked list

4. Wait – process stays in blocked while others execute

5. Unblock – at the start of the next cycle, the oldest blocked process is moved back to the tail of its original priority queue

6. Re-execution – process is now eligible to run; if it still needs DISC, it will not be blocked again (flag already set)

7. Completion – cycles reach 0 → process terminates

5**. Performance Bottleneck & Theoretical Improvement**

Main bottleneck: Global starvation counter and strict FIFO queues – a burst of HIGH jobs can still delay MEDIUM/LOW for up to 5 cycles, and aging is not supported.

Theoretical improvement – Aging with Multi-Level Feedback Queue (MLFQ):

- Add priority boost bit: increment a wait-counter each cycle a process is skipped
- When counter > threshold, promote the process one level (e.g., LOW → MEDIUM)
- Reset counter after promotion
- Complexity still O(1) per operation, but starvation bound becomes 1 cycle instead of 5
- Throughput of HIGH jobs remains high, latency of LOW jobs drops significantly

Conclusion:

The manual linked-list scheduler is simple, fair, and efficient for the proposed scenario.

The anti-starvation rule and “disc” blocking mechanism ensure correctness and resource safety.

Aging-enhanced MLFQ is the next evolutionary step for production-grade systems.