Project Description: Simple Paging-Based Memory Management Simulator

Group Members	Roll No
Aqiba Abdul Qadir	CS-22003
Tooba Aftab	CS-22020
Bareera	CS-22030

Project Description:

This simulation models how an operating system manages memory allocation using **paging** with a First-Come-First-Served (FCFS) scheduling approach. It provides a clear visualization of how processes acquire and release memory frames while maintaining efficient system performance. The use of paging indicates real-world functionality of modern-world systems. Paging eliminates internal fragmentation and substantially reduces external fragmentation. Implementation of simple paging is also comprehensible and intuitive to code.

Core Functionality

1. **Memory Allocation Strategy**: Physical memory is partitioned into fixed-size frames (configurable via FRAMESIZE). Each process's memory requirements are converted into page counts, with the system tracking both contiguous and fragmented allocations.
2. **Process Lifecycle Management**: Processes are characterized by: Arrival time (when they request memory), Memory footprint (converted to required pages), Execution duration. A priority queue (min-heap) determines which process exits next, enabling automatic memory reclamation.
3. **Resource Contention Handling**: If memory is insufficient for a new process: The earliest-completing process is identified **(O(log n) time via heap).** Its memory is released. The new process acquires the freed frames. This prevents deadlock while maximizing memory utilization.

Technical Strengths

✔ **Optimized Data Structures**: Min-heap ensures O(log n) insertion/removal when managing process completion times. Efficient frame tracking via arrays with O(1) access

✔ **Real-World Behavior Simulation**: Accurately models: External fragmentation. Process deallocation based on execution timelines. Memory reuse patterns.

✔ **Interactive Debugging**: Step-through execution allows observing memory state transitions. Clear terminal visualization of frame allocations as process is being read from the input file.

Assumptions

- **RAMSIZE** = 256 KB
- **FRAMESIZE** = 4 KB
- ⇒ **totalFrames** = 256 / 4 = 64 frames
- **MAXCAPACITY** = 15 processes; considered 10 processes while creating input file.
- We assume that the single process size will always be smaller than the entire RAM as simple paging requires full process allocation to RAM.
- Maximum size of processes was considered 200KB while minimum was 20KB. Average size is 110KB.
- -1 in RAM represent empty spaces.

This “real-time” simulation perfectly illustrates simple paging under FCFS: each new process either “just fits,” or else the oldest-finishing jobs get evicted until enough free frames exist, then the new job is loaded in circular fashion.

Folder Structure

```
OS_OEL/
├── headers/
│   ├── paging.h
│   ├── heap.h
│   └── table.h
├── main.c
├── heap.c
├── table.c
├── input.txt
└── readme.md
```

The input.txt file contains 30 lines where each 3 lines determine the arrival time, size in KBS and execution time in milliseconds of the processes respectively.

Conclusion

By combining efficient data structures with clear visualization, this simulator effectively demonstrates core memory management principles. The FCFS approach provides predictable behavior while the heap-based scheduling ensures optimal resource recovery. Future extensions could transform this into a more complex variable paging system.