COEN383
Advanced Operating System
Project 4 Report - Swapping and Paging Simulator
Group No. #1

(Gaurav Gaikwad, Gauri Dave, Wineel Wilson Dasari, Anthony Bryson, Dhanashree Bharuka)

Introduction

The project simulates memory management techniques through page replacement algorithm implementations. The system displays how operating systems operate paging functions when memory resources are limited. The program executes processes with multiple memory usage levels while testing five different page replacement methods which include FCFS, LRU, LFU, MFU and Random. C programming with linked lists for memory pages and arrays for process queues serves to achieve this functionality.

System Design and Implementation

Memory and Process Configuration

- **Total Physical Memory:** 100 MB
- **Page Size:** 1 MB (100 pages)
- **Processes:** 150 total, with sizes of 5, 11, 17, or 31 MB
- Each process has a random arrival time (0 - 59 seconds) and random service duration (1 - 5 seconds)

Page Replacement Algorithms

- **FCFS:** Evicts the page brought earliest
- **LRU:** Evicts the least recently used page
- **LFU:** Evicts the least frequently used page
- **MFU:** Evicts the most frequently used page
- **Random:** Evicts a random page from memory

Memory Reference Behavior

- Each process starts execution on page 0
- Every 100ms, it makes a memory reference
- Locality of reference is applied (70% chance to reference nearby pages)

Modules and File Structure

File Name	Usage
SIMULATION.c	Controls simulation execution and timing
FCFS.c	Implements FCFS page replacement algorithm
LRU.c	Implements LRU algorithm
LFU.c	Implements LFU algorithm
MFU.c	Implements MFU algorithm
R.c	Implements Random page replacement algorithm
PAGE.c / PAGE.h	Defines and manages page structures and logic
README.md	Project documentation and instructions
Makefile	Automates compilation of the project
output/-	Saved output logs

Simulation and Page Management

Process Queue and Job Scheduling
- Processes are generated and sorted based on arrival time
- A job is swapped in only if 4 or more free pages are available

Paging and Reference Handling
- Every 100ms, processes generate page references
- Pages not in memory trigger a page-in operation
- Page-in may trigger a page replacement decision using the selected algorithm

Statistics and Output
- For each memory reference: timestamp, PID, page referenced, in-memory status, eviction if needed
- At the end: average number of processes swapped in over 5 runs

Compilation and Execution

Using Make

```
cd src
make
```

Executing

```
./simulation FCFS
./simulation LRU
./simulation LFU
./simulation MFU
./simulation Random
```

Saving Output

```
./simulation FCFS > ../output/FCFS.txt
./simulation LRU > ../output/LRU.txt
./simulation LFU > ../output/LFU.txt
./simulation MFU > ../output/MFU.txt
./simulation Random > ../output/Random.txt
```

Cleaning Build

```
make clean
```

Sample Output

Each run prints:

- Page reference logs per process
- Eviction decisions
- Process entry and exit messages
- Final summary of average swapped-in processes

Challenges and Learnings

Memory Reference Timing:
Simulating references every 100ms per process required fine-tuned nested loops and precise timestamp tracking.

Eviction Logic:
Each algorithm needed precise implementation and testing to ensure correct victim page selection under all conditions.

Simulation Control:
Handling 150 processes and ensuring correct page tracking, reuse, and eviction across multiple runs added considerable complexity.

Output Management:
High verbosity during the simulation runs made output redirection and debug flag tuning necessary.

Conclusion:

The project delivered applied experience in paging techniques and memory administration systems. The system facilitated understanding of operating system internals through simulations of page faults together with memory access behaviors and replacement methods across different algorithms. The system remains reliable through its use of reproducible simulations together with algorithm switching logic and modular C code design.