NED University of Engineering and Technology
Department of Computer and Information
Systems Engineering

Open Ended Lab
CS-329 Operating Systems

Submitted to:

Dr. Zareen Sadiq

Prepared by:

Manahil Ijaz(CS-22011)
Mahwish Hussain(CS-22016)
Neha Nauman Khan(CS-22024)

Paging Memory Management Report

1. Introduction

This document outlines the functions, flow, and internal structures used in a memory paging simulation. The simulation demonstrates how operating systems allocate, manage, and swap memory pages between RAM and Virtual Memory (VM) for multiple processes. The system ensures optimized page allocation, avoids fragmentation, and facilitates smooth swapping of leftover pages.

2. System Overview

The system simulates:

- **Memory Allocation**: Efficient initial and dynamic memory page allocation for processes.
- **Memory Deallocation**: Controlled swapping of remaining pages after initial allocations.
- **Paging Process Simulation**: Imitates real-world operating system behavior using structures and defined logic.
- **Key Components**: RAM, VM, framesAllocated[], pgRemainingList[], and the processes[] array.

3. Core Functions and Their Purpose

3.1 Function: newTOReadyOrReadySuspend(int start, int limit, struct pgDict MEM[], int processNo)

Purpose:
Allocates a specific number of memory pages (limit) for a process (identified by processNo), starting at a given index (start) in the memory array MEM[] (representing RAM or VM).

Detailed Explanation of limit Calculation:

- **Fixed Allocation Start**:
  fixedStart = (i-1) * pgToAllocate;
  Represents the theoretical starting position for a process assuming equal frame

distribution.

- **Adjustment Scenarios**:
    - If fixedStart > start_Process:
      Some pages were skipped previously; adjust limit to close the gap and continue sequentially.
    - Else:
      Normal allocation of pgToAllocate pages.

Benefits:

- Minimizes memory fragmentation.
- Ensures continuous page placement.
- Optimizes memory management for swapping and future operations.

3.2 Function: assigningLeftOverPgs()

Purpose:
Handles the gradual swapping of unallocated process pages into RAM after the initial allocations.

Working:

- Iterates over all processes.
- If pgRemainingList[i] == 0, no action is needed ("No page to swap").
- Otherwise:
    - Calculates replace_index using circular replacement logic.
    - Determines new page number new_pg.
    - Updates memory frame (memRepresentation) with:
        - pgNO
        - processNo

        - arrivalTime

    - Displays updated memory layout using viewMem() after every swap.

Key Variables:

- replace_index: Target memory frame for replacement.
- allocatedFrames: Initially allocated frames for each process.
- swappedPg: The page being swapped into memory.

Importance:

- Ensures that all process pages are eventually loaded into RAM.
- Gradual loading avoids memory contention and improves overall system stability.

3.3 Function: readFromFile()

Purpose:
Reads process information from an external file processFile.txt.

Working:

- Populates processes[] with:
    - pid (Process ID)
    - size (Process size in bytes)
    - arriveTime (Process arrival time)

Structure Used:

```
struct process {
   int pid;
   int size;
   int arriveTime;
};
```

Significance:

- Provides real-world process information dynamically.
- Supports flexible simulation with varying process attributes.

3.4 Function: main()

Purpose:
The central driver that orchestrates the paging simulation.

Execution Flow:

1. **Data Loading**:
   Calls readFromFile() to load process data.
2. **Initialization**: Sets RAM and VM arrays as empty.
3. **Page Calculation**: Divides process size by page size to determine the required number of pages.
4. **Frame Allocation**: Calculates pgToAllocate (equal division of RAM frames among processes).
5. **Process Handling**: For each process:
   - Records memory frame starting index.
   - Allocates pages based on their needs:
     - Full allocation if pages ≤ pgToAllocate.
     - Partial allocation otherwise with careful limit management.
   - Updates framesAllocated structure.
   - Records leftover pages in pgRemainingList[].
   - Displays memory state after each allocation via viewMem().
6. **Swapping Remaining Pages**: Calls assigningLeftOverPgs() to gradually load all remaining pages into RAM.

4. Important Data Structures

4.1 Structure: framesAllocated

Attributes:

- int processId: Process ID.
- int StartingPgNo: First page number.
- int EndingPgNo: Last page number allocated initially.
- int StartingMemFrame: Memory frame index where the process's first page is stored.

Purpose:

- Tracks page allocations and memory positions.
- Aids efficient page swapping and visualization.
- Prevents overwriting during dynamic memory operations.

Features and Challenges

The system minimizes fragmentation through smart limit calculations during page allocation and supports dynamic, efficient swapping of pending pages. Real-time memory visualization after each operation enhances transparency. Key challenges included avoiding fragmentation, addressed by adjusting limits based on allocation gaps, and managing leftover pages without disturbing existing frames, solved through circular replacement and careful tracking via framesAllocated[]. Realistic arrival and paging behavior were achieved by structured file-based data loading.