FACULTY OF COMPUTING

SEMESTER 2 2024/2025

SECR2043 – OPERATING SYSTEMS

SECTION 2

PROJECT REPORT

TOPIC: PAGING CONCEPT

LECTURER: DR. ZAFRAN BIN MUHAMMAD ZALY SHAH

Video Link: https://youtu.be/39qxF00_44M

GROUP Quadratech

STUDENT NAME	MATRIC NO
CHUA JIA LIN	A23CS0069
EVELYN GOH YUAN QI	A23CS0222
YASMIN BATRISYIA BINTI ZAHIRUDDIN	A23CS0201
NABIL AFLAH BOO BINTI MOHD YOSUF BOO YONG CHONG	A23CS0252

TABLE OF CONTENTS

PAGING CONCEPT VISUALIZATION THROUGH EMERGENCY PATIENT TRANSFER

1.0 ABSTRACT

This project explores the concept of paging in operating systems through a real-world analogy of hospital ward and ICU room management. Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory, thereby minimizing fragmentation and improving system efficiency. In our simulation, patients represent processes, general wards symbolize pages in virtual memory, and ICU rooms act as frames in physical memory. The system allocates ICU beds to patients based on availability, reflecting how pages are loaded into frames during program execution. A fixed-size page table is implemented to map general wards to ICU rooms, while also highlighting the occurrence of internal fragmentation when ICU beds remain partially unutilized. By applying this model, the project aims to simplify and illustrate how the paging mechanism functions under constrained memory conditions. The simulation not only serves as an educational tool but also bridges the gap between abstract OS concepts and real-world scenarios, especially in resource-constrained environments like healthcare systems.

2.0 INTRODUCTION

Effective memory management is necessary for modern computer systems to operate at their best, particularly when resources are limited. This study proposes a hospital ward and intensive care unit allocation model based on operating system paging principles in order to represent and better understand these concepts. The analogy involves general wards acting as pages, ICU rooms as frames, and patients as the processes requiring memory allocation. Each general ward consists of a fixed number of beds, and only a limited number of ICU rooms are available to handle critical patients, much like limited physical memory frames in a computer system.

This project uses a fixed-size page table to construct a Python simulation that assigns patients to general wards and transfers specific wards to intensive care units. Additionally, it presents the idea of internal fragmentation, in which the number of patients and the available space in designated intensive care units may not match, resulting in empty beds. By using this model, the project makes abstract memory management principles more concrete and understandable by illustrating the real-world effects of inefficient memory consumption in addition to visualizing how paging operates.

3.0 PROBLEM STATEMENT

The problem of critical patient data being loaded and retrieved properly in a hospital with limited memory has been solved here. The scenario considers a hospital system where a few ICU beds are allocated and can hold only a certain number of patient records at a time. Every patient is a page in the memory frames which is ICU bed.

The problem occurs when more patients need immediate access, and the system can now take in no more records because it is at the limit of its page table size. This is how precise memory control could deny access regardless of whether there are unused memory frames and hence become detrimental in critical situations. This simulation is meant to help illustrate and clarify the working of the Paging Concept in systems such as these with limited resources. This shows the effect of having no replacement policy, which disallows any further pages from being loaded even though there are other memory resources accessible after a tracking limit is reached.

We expect this simulation will assist students understand the importance of memory tracking, page faults, and dynamic paging techniques. The application aims to improve learning by connecting fundamentals of operating system ideas to actual healthcare scenarios.

4.0 RELATED WORKS

The paging concept in operating systems is used to manage memory efficiently by dividing the memory into smaller fixed-size units called pages (in virtual memory) and frames (in main memory). To help understand this, we used a real-world hospital scenario to explain the paging concept. In this project, we compared the general ward to virtual memory and the ICU rooms to main memory. Each patient represents a process, each ward acts as a page, and each ICU room is like a frame in memory. When a patient (process) needs emergency care, they are transferred from the general ward (page) to the ICU room (frame) based on a medical instruction list, just like how pages are loaded into main memory using a page table. This scenario clearly shows how the paging concept works by illustrating how patients (processes) are moved between different memory levels based on need and availability.

5.0 METHODOLOGY

This project followed a structured methodology divided into four main phases: Planning, Design, Implementation, and Testing & Evaluation. The approach aimed to translate the abstract concept of paging into a real-world hospital scenario that can be easily visualized and understood.

5.1 Planning Phase

In this initial phase, the concept and analogy for the simulation were defined. The paging concept in operating systems was mapped to a hospital scenario, where:

- General wards represent virtual memory pages
- ICU rooms represent physical memory frames
- Patients represent processes

The goal was to demonstrate how a limited number of ICU rooms (frames) could accommodate only a subset of patients (processes) from general wards (pages), similar to how pages are loaded into memory frames during execution. It is also identified that a fixed-size page table would be used without any page replacement mechanism, to simulate memory constraints.

5.2 Design Phase

During this phase, the structure of the simulation was planned:

- Ward Setup: 6 general wards (pages), each with 4 patient beds.
- ICU Setup: 3 ICU rooms (frames), each with a capacity of 4 beds.
- Page Table: Limited to 3 entries to simulate restricted physical memory access.
- Allocation Logic: A mapping would be created where only 3 wards are actively assigned to ICU rooms.

A strategy was also developed to account for internal fragmentation — the number of unused beds in each ICU room, reflecting wasted memory space.

5.3 Implementation Phase

In this phase, the simulation was developed using Python. The following components were implemented:

- A dictionary to hold patient counts per ward
- A page table that maps specific wards to ICU rooms
- A function to allocate patients from wards to ICU rooms based on page table entries
- Internal fragmentation tracking to record unused ICU beds
- Unique patient identifiers to indicate the source ward

No page replacement algorithms were included, aligning with the initial goal of modeling a static paging system under memory constraints.

5.4 Testing & Evaluation Phase

The final phase involved running the simulation and evaluating the output:

- The ward distribution and ICU allocations were printed
- ICU room utilization and fragmentation were calculated
- The total internal fragmentation was summed to assess efficiency

The output confirmed the simulation correctly modeled paging behavior and internal fragmentation, making the abstract OS concept tangible and educational. The results helped highlight the importance of efficient memory allocation and the limitations of fixed-size page tables.

6.0 DESIGN & IMPLEMENTATION

By representing wards as pages, ICU rooms as memory frames, and patient allocations as memory usage, the hospital ward and ICU allocation scenario was systematically converted into a Python program throughout this phase. To ensure an accurate simulation of a paging system in an operating system context, the design included logic to handle bed allocations, a fixed-size page table to simulate limited ICU mappings, and data structures to represent patient distribution and internal fragmentation calculation.

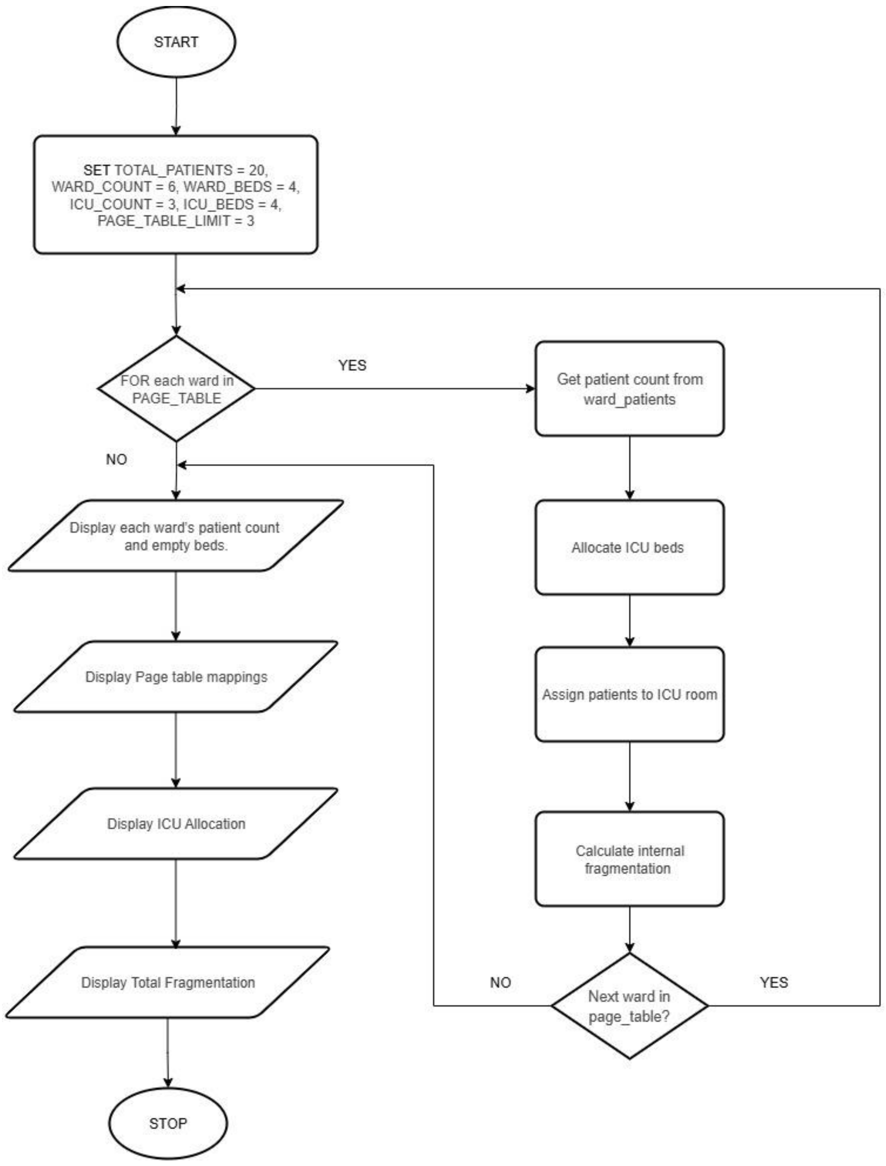
6.1 Design Phase

The core components of the design included:

1. Ward Structure (Pages):
   - Six general wards were defined, each containing four beds.
2. ICU Rooms (Frames):
   - Three ICU rooms, each with four beds, were used to represent fixed-size memory frames.
3. Page Table Design:
   - A fixed-size page table with only three entries was created to map specific wards (pages) to ICU rooms (frames).
   - This limitation was intentional to mimic real-world memory constraints in operating systems.
4. Patient Allocation Logic:
   - A function was planned to allocate patients from mapped wards into ICU rooms, ensuring the number of beds per room was not exceeded.
   - Any unfilled beds in ICU rooms were flagged as internal fragmentation.

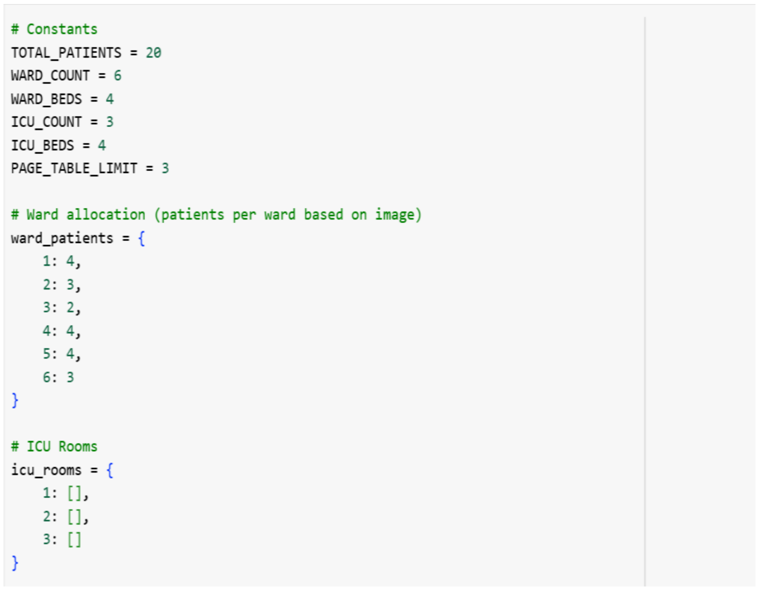
5. Internal Fragmentation Calculation:
   - A mechanism was included in the design to calculate and report unused beds in each ICU room, representing memory waste within allocated blocks.

6. System flowchart


START
SET TOTAL_PATIENTS = 20, WARD_COUNT = 6, WARD_BEDS = 4, ICU_COUNT = 3, ICU_BEDS = 4, PAGE_TABLE_LIMIT = 3
FOR each ward in PAGE_TABLE
YES
NO
Get patient count from ward_patients
Allocate ICU beds
Assign patients to ICU room
Calculate internal fragmentation
Next ward in page_table?
NO
YES
Display each ward's patient count and empty beds.
Display Page table mappings
Display ICU Allocation
Display Total Fragmentation
STOP

6.2 Implementation

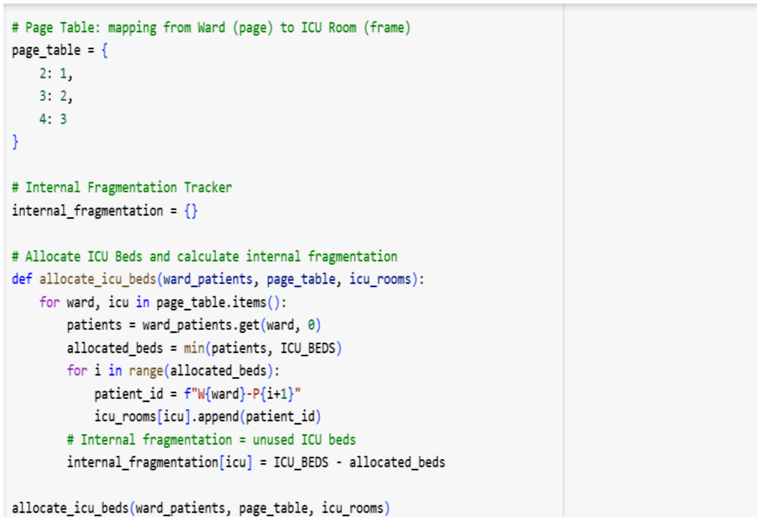
To implement this project, we code in Python. Below are the codes:



```python
# Constants
TOTAL_PATIENTS = 20
WARD_COUNT = 6
WARD_BEDS = 4
ICU_COUNT = 3
ICU_BEDS = 4
PAGE_TABLE_LIMIT = 3

# Ward allocation (patients per ward based on image)
ward_patients = {
    1: 4,
    2: 3,
    3: 2,
    4: 4,
    5: 4,
    6: 3
}

# ICU Rooms
icu_rooms = {
    1: [],
    2: [],
    3: []
}
```

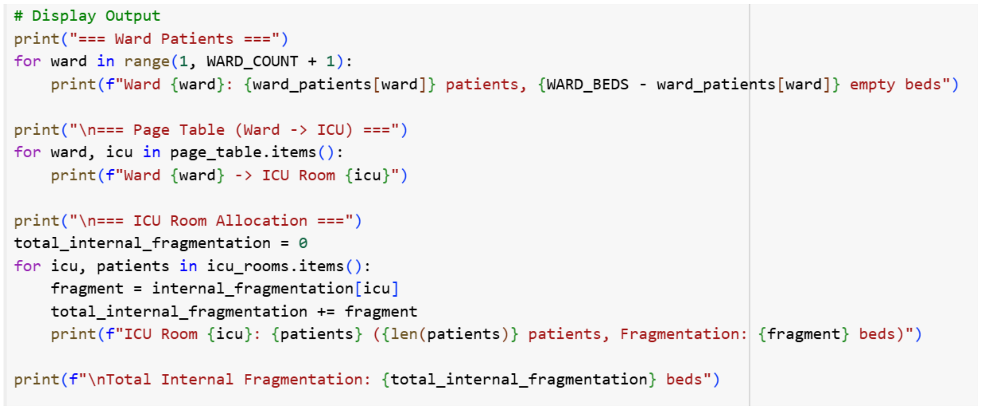


```python
# Page Table: mapping from Ward (page) to ICU Room (frame)
page_table = {
    2: 1,
    3: 2,
    4: 3
}

# Internal Fragmentation Tracker
internal_fragmentation = {}

# Allocate ICU Beds and calculate internal fragmentation
def allocate_icu_beds(ward_patients, page_table, icu_rooms):
    for ward, icu in page_table.items():
        patients = ward_patients.get(ward, 0)
        allocated_beds = min(patients, ICU_BEDS)
        for i in range(allocated_beds):
            patient_id = f"W{ward}-P{i+1}"
            icu_rooms[icu].append(patient_id)
        # Internal fragmentation = unused ICU beds
        internal_fragmentation[icu] = ICU_BEDS - allocated_beds

allocate_icu_beds(ward_patients, page_table, icu_rooms)
```



```python
# Display Output
print("=== Ward Patients ===")
for ward in range(1, WARD_COUNT + 1):
    print(f"Ward {ward}: {ward_patients[ward]} patients, {WARD_BEDS - ward_patients[ward]} empty beds")

print("\n=== Page Table (Ward -> ICU) ===")
for ward, icu in page_table.items():
    print(f"Ward {ward} -> ICU Room {icu}")

print("\n=== ICU Room Allocation ===")
total_internal_fragmentation = 0
for icu, patients in icu_rooms.items():
    fragment = internal_fragmentation[icu]
    total_internal_fragmentation += fragment
    print(f"ICU Room {icu}: {patients} ({len(patients)} patients, Fragmentation: {fragment} beds)")

print(f"\nTotal Internal Fragmentation: {total_internal_fragmentation} beds")
```

- Step 1: Initialize all the constant
  - Constants such as the total number of patients (TOTAL_PATIENTS), total number of wards (WARD_COUNT), number of beds in each ward (WARD_BEDS), ICU room count (ICU_COUNT), ICU beds per room (ICU_BEDS), and the page table limit (PAGE_TABLE_LIMIT) are defined.
  - These constants represent the static configuration of the hospital's ward and ICU setup
- Step 2: Allocate patients per ward
  - ward_patients is created to represent the number of patients in each ward.
  - Each ward is assigned a key (1 to 6), and the corresponding value is the number of patients in that ward.
- Step 3: Mapping from Ward (page) to ICU Room (frame)
  - Page table maps logical addresses (wards) to physical addresses (ICU rooms).
  - The dictionary page_table establishes which ward is linked to which ICU room.
  - Only three entries are allowed due to the PAGE_TABLE_LIMIT, reflecting a real-world constraint such as limited ICU mapping or access control.
- Step 4: Set the Internal Fragmentation Tracker
  - internal_fragmentation is initialized to track the number of unused ICU beds (internal fragmentation) in each ICU room.
- Step 5: Allocate ICU Beds and calculate internal fragmentation

  A function allocate_icu_beds is defined and called. It loops through the wards that are mapped to ICU rooms via the page_table. For each ward:
  - It retrieves the number of patients.
  - It allocates a maximum of ICU_BEDS beds (up to 4), based on how many patients there are.
  - It records which patient from which ward occupies which ICU bed.
  - It calculates internal fragmentation by subtracting the number of allocated beds from the total ICU capacity (i.e., ICU_BEDS - allocated_beds)
- Step 6: Display output

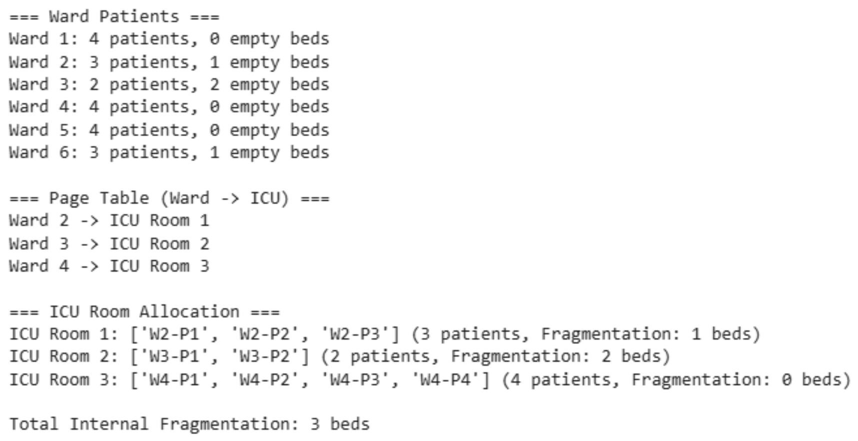
  The code prints out
  - The number of patients in each ward and the number of empty beds.
  - The mapping of wards to ICU rooms (page table).

- ICU room allocations including the list of patients in each ICU, the number of patients per ICU, and how many beds were wasted (internal fragmentation).
- The total internal fragmentation across all ICU rooms.

7.0 RESULTS & DISCUSSION

7.1 RESULTS

The simulation portrays a scenario in which paging is utilized to manage patient files between general wards and ICU rooms under a constrained memory environment. There are 6 general wards, each with a maximum number of 4 beds, and a varying number of patients. The system thus simulates the loading of wards into ICU rooms, wherein each ICU room can host up to 4 patients, serving as memory frames.



```
=== Ward Patients ===
Ward 1: 4 patients, 0 empty beds
Ward 2: 3 patients, 1 empty beds
Ward 3: 2 patients, 2 empty beds
Ward 4: 4 patients, 0 empty beds
Ward 5: 4 patients, 0 empty beds
Ward 6: 3 patients, 1 empty beds

=== Page Table (Ward -> ICU) ===
Ward 2 -> ICU Room 1
Ward 3 -> ICU Room 2
Ward 4 -> ICU Room 3

=== ICU Room Allocation ===
ICU Room 1: ['W2-P1', 'W2-P2', 'W2-P3'] (3 patients, Fragmentation: 1 beds)
ICU Room 2: ['W3-P1', 'W3-P2'] (2 patients, Fragmentation: 2 beds)
ICU Room 3: ['W4-P1', 'W4-P2', 'W4-P3', 'W4-P4'] (4 patients, Fragmentation: 0 beds)

Total Internal Fragmentation: 3 beds
```

Results of Paging Simulation in Emerging Patient Transfer

The results show 4 patients in Ward 1, Ward 4, and Ward 5, 3 patients in each of Ward 2 and Ward 6, and there are 2 patients in Ward 3. From the page table given, only 3 wards can be tracked and mapped into ICU rooms because of the limited size of the table. The wards mapped are Ward 2 in ICU Room 1, Ward 3 in ICU Room 2, and Ward 4 in ICU Room 3. This type of mapping exhibits the working of paging where only those pages (wards) are brought into memory (ICU), which is necessary.

The ICU room allocation shows 3 patients of Ward 2 in ICU Room 1, leading to one bed being idle as a case of internal fragmentation. ICU Room 2 holds only 2 patients in Ward 3, leaving 2 beds unused. ICU Room 3, assigned to Ward 4, is fully utilized, as all 4 beds are occupied. The internal fragmentation of all ICU rooms, being 3 beds, is a case of memory space being left unused due to uneven distribution of patient per ward.

7.2 DISCUSSION

The presence of inefficiencies is creating a situation where the data groups are of variable sizes, with wards having various numbers ranging from patients that get allocated fixed-size frames or ICU rooms. Paging selective access to hospital data would be just like an actual operating system loading only the necessary pages into the memory. But the limitations of the page table restrict how many wards may be kept as ICU at a time, thus may hinder timely retrieval of patient records whenever many wards need an ICU.

Internal fragmentation is the main lesson to learn from this simulation. Although ICU Room 3 does make use of all the beds provided, Room 1 and 2 are not sufficiently utilized. This translates to a waste of physical memory space, which is the way that disparate allocation sizes bring down the performance of the system. Three beds complete internal fragmentation is an example of a common paging scenario in which the predetermined frame size does not match the real data requirements.

By situating the memory management issue within a hospital setting, the entire simulation provides a user-friendly demonstration. It teaches that the effectiveness of the system's allocation affects its performance in addition to the paging limit's effect on access. These revelations contribute to a deeper comprehension of memory concepts from a theoretical and practical standpoint.

8.0 FUTURE WORK & CONCLUSION

8.1 FUTURE WORK

The current hospital paging simulation helps us understand how memory paging works by using a real-world example. Patients are treated as processes, wards as pages, and ICU rooms as frames. This model shows how patients are moved to ICU beds based on memory availability, similar to how pages are loaded into memory frames.

The future work will focus on improving this simulation in several ways:

- **Page Replacement Algorithms**
  Add other replacement methods such as Least Recently Used (LRU) or Optimal to better manage ICU room allocation, especially when space is limited.
- **Dynamic Patient Access**
  Let users enter their own access patterns to test how the hospital system handles different patient (process) behaviors.
- **Visual ICU Mapping**
  Create a simple visual interface to show which patients are in which ICU beds and how they change over time.
- **Custom Settings**
  Allow users to change the number of patients, ICU beds, and ward sizes to explore different situations.

8.2 CONCLUSION

In conclusion, this project provides a clear and simple way to learn about the paging concept in operating systems. By using a hospital scenario, it becomes easier to understand how pages are moved into memory frames and how replacement is handled when space runs out. The future improvements will make the simulator even more useful and interactive for real-life understanding.

9.0 ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our esteemed lecturer, Dr. Zafran Bin Muhammad Zaly Shah, for his guidance, support, and insightful advice throughout the development of this project. Dr Zafran's clear explanations and helpful feedback made it easier for us to understand the paging concept using a real-world hospital scenario. Besides that, we also thank our course mates and friends who provided valuable suggestions and helped us to test the simulator. Their effort played an important role in improving the results. Lastly, we are thankful for the online learning resources and references that supported us during the development of this project.

10.0 REFERENCES

GeeksforGeeks. (2025, May 22). *Paging in operating system*. GeeksforGeeks. https://www.geeksforgeeks.org/operating-systems/paging-in-operating-system/

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.

Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson.

The_Anshuman. (2024, January 10). What is paging in OS? - The_Anshuman - Medium. *Medium*. https://medium.com/@The_Anshuman/what-is-paging-in-os-4b9101bee8c1

Video Link: https://youtu.be/39qxF00_44M