# Report for the Operating System Assignment 3


## By

## Ezaz Ahmad


## Date: 19/10/2023

## 1. Testing Approach and Methodology

In building my program to simulate paging with virtual memory, I applied a structured testing approach to ensure the robustness and reliability of my solution. The methodology followed:

Unit Testing: Every class and method (Process, Memory, Page, and Simulator) was tested in isolation. Mock data was generated to verify the correctness of each function.

Integration Testing: Once individual units were verified, I combined them and checked how they interacted, especially the interactions between the paging mechanism and the scheduler.

Scenario-based Testing: Using the provided sample data sets, I recreated specific scenarios in the assignment brief. Additionally, I created more diverse scenarios with varying numbers of processes, page requests, and memory frames to further challenge my program.

## 2. Comparison of Page Replacement Methods

Based on the results from the simulations:

Clock with Fixed Allocation & Local Page Replacement: This method offered predictable outcomes. Processes were confined to their allocated frames, ensuring that no process could monopolize the memory. This resulted in relatively balanced turnaround times across processes.

Clock with Variable Allocation & Global Page Replacement: This method showcased more dynamic behavior. Processes weren't confined to any specific frames, leading to cases where a process could potentially use up more memory. While this sometimes resulted in faster completions for some processes, it also sometimes led to longer waiting times for others.

Observations: The Variable Allocation method seemed more efficient when the total number of page requests from all processes was close to or less than the total frames available. In contrast, Fixed Allocation ensured fair distribution but sometimes wasn't the most optimal in terms of total execution time.

## 3. Edge Cases and Behaviors

Several edge cases were considered during development, for example:

- Single Process in Memory: When only one process is in memory, how does the system behave? Does it still use the page replacement algorithm or just load everything sequentially?
- All Processes Requesting the Same Page: What happens if, by coincidence, every process in a cycle needs the same page?
- High Page Fault Rate: If every page requested leads to a page fault, how does the system cope?

For these edge cases, the algorithms demonstrated robustness. Especially for the third scenario, the Global Replacement strategy, in some instances, performed swaps more frequently than its Fixed counterpart, causing slight overhead.

## 4. Review of Results and Observations

While developing the simulation, several interesting behaviors and challenges I noted, for example:

**Simultaneous Page Faults:** Handling multiple page faults occurring at the same time was initially a challenge.

**Unexpected Behavior in Variable Allocation:** Initially, I noticed some processes completing far quicker than expected, while others took significantly longer. This was due to the global nature of the page replacement where some processes were, unintentionally, prioritized over others.

**Challenge in Clock Algorithm:** Implementing the Clock algorithm, especially differentiating between the two replacement policies, was intricate. Ensuring that the pointer moved correctly and that pages were accurately replaced required careful debugging.

In conclusion, this project provided deep insights into the workings of page replacement algorithms and their real-world implications. The Clock algorithm, with its two variants, showcases the trade-offs systems designers need to consider: fairness versus optimal performance. The balance between the two is contingent upon the specific needs and constraints of the system in question.