TARLAC STATE UNIVERSITY
COLLEGE OF COMPUTER STUDIES
COMPUTER PROGRAMMING 2 – CC3
2nd Semester – A.Y. 2024-2025

Simulation and Analysis of a Page-Replacement Algorithm in a Virtual Memory System

A Project

Submitted to the Faculty of the College of Computer Studies
San Isidro Campus, Tarlac City,

Tarlac State University

In Partial Fulfillment of the Requirements for the Subject

Operating Systems

Academic Year 2024-2025

Submitted By:
Quiambao, Eric Janssen P.

Jo Anne Cura
Subject Instructor

Date of submission: May 21 , 2025

Table of Contents

Project Overview

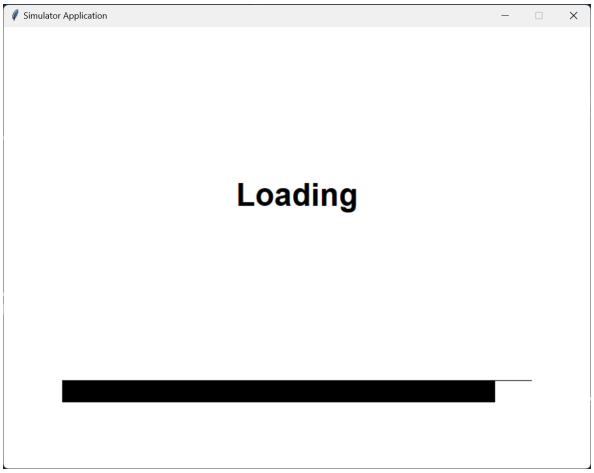
This simulator is an educational Python program developed using Tkinter that illustrates three traditional page-replacement schemes—FIFO, LRU, and OPT—via an interactive GUI. Besides fulfilling the course requirement, its other purpose is to give students a feel of how various algorithms handle virtual memory frames under different workloads. By creating a random reference string each time it is run and graphically illustrating page faults, the program makes theoretical principles tangible.

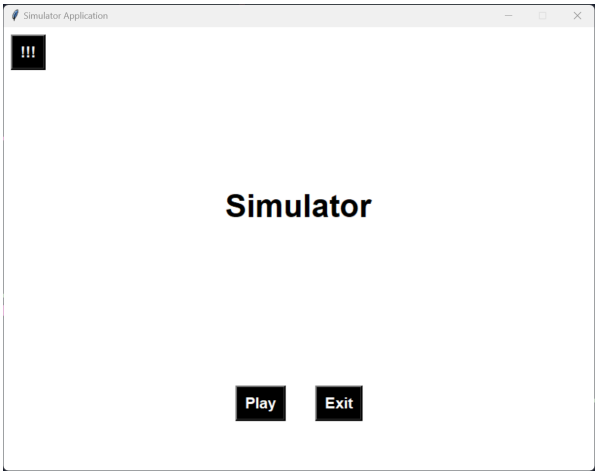
Under the hood, the project uses Python 3.11 for fast development and readability, the standard Tkinter library for basic yet efficient animations, and the random module for uniform sequences of integers. A PyInstaller build bundles the interpreter and dependencies into a single Windows executable for easy distribution, albeit requiring resource path and Tkinter asset handling. The random generator now selects integers 0–9 for a length between 1 and 30, which can be extended with locality-biased models to simulate real-world access patterns.

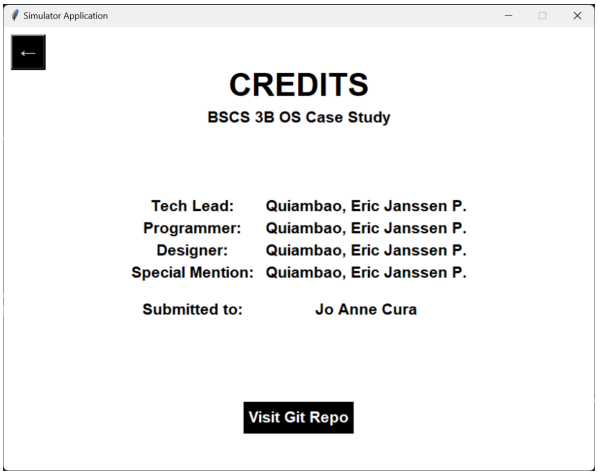
The GUI is divided into four frames for an uninterrupted user experience. The LoadingFrame reassures the user at startup, the MainMenuFrame provides navigation to simulate or quit and view credits, the CreditsFrame thanks contributors and references the GitHub repository, and the SimulatorFrame offers the primary functionality—choosing an algorithm, adjusting parameters, and observing frames update in real time. The link to the repo is this https://github.com/Ensues/OS-Case-Study or scan the QR code below

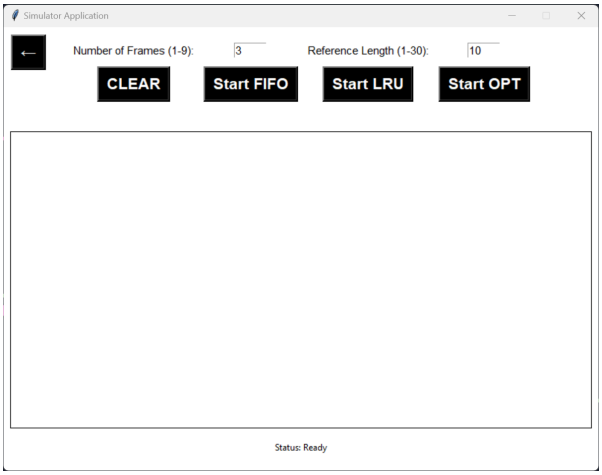
Documentation

My application is organized into four distinct frames, each serving a dedicated role in the user workflow. Together, they create a cohesive interface that guides users from the main menu through simulation setup, execution, and result review. Below is a concise overview of each frame and its primary responsibilities.



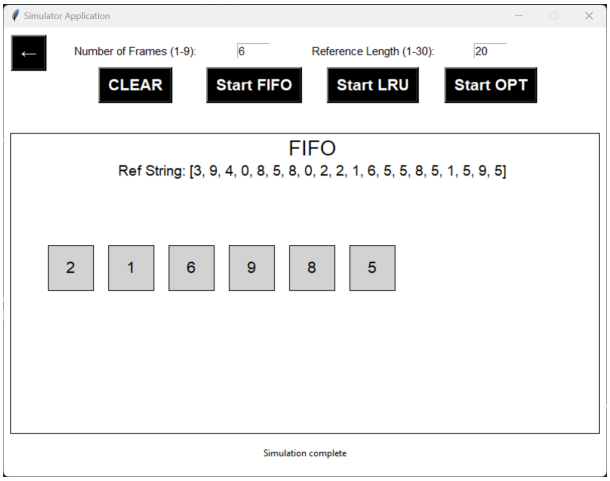


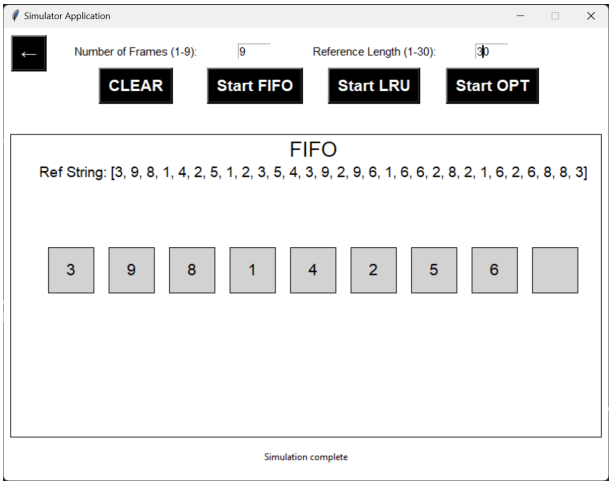




FIFO Output

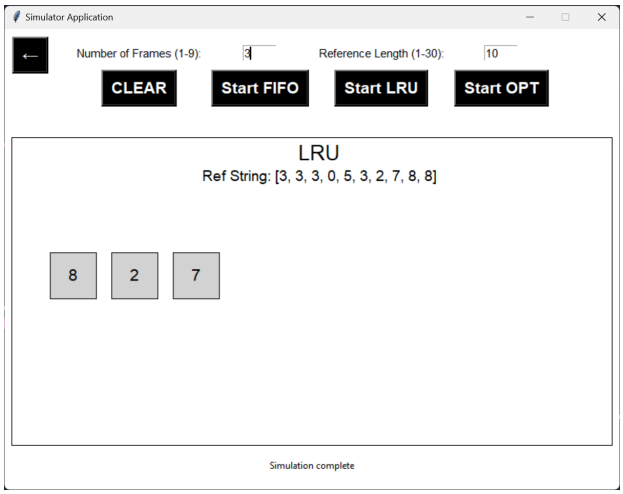
The FIFO algorithm manages pages in a strict queue order by evicting the page that was loaded earliest whenever a page fault occurs. In our demonstration we ran three example scenarios: 3 frames with reference string length 10, 6 frames with length 20, and 9 frames with length 20. Each screen capture walks through every step from an empty frame set to the final contents so you can observe how FIFO pushes out the oldest page at each fault. The same three configurations are used for the LRU and OPT algorithms to ensure a consistent basis for comparison.

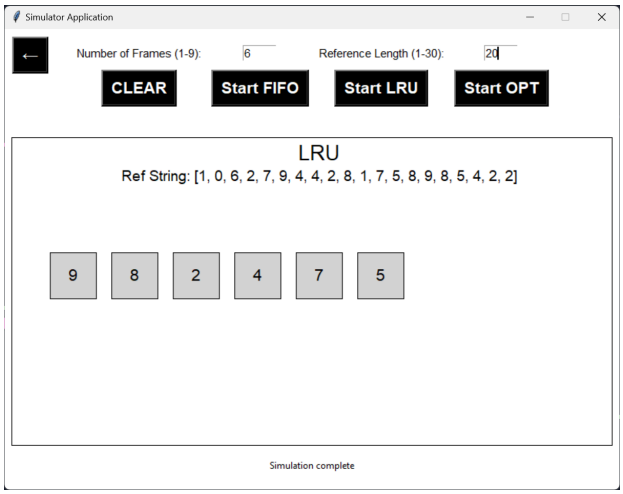


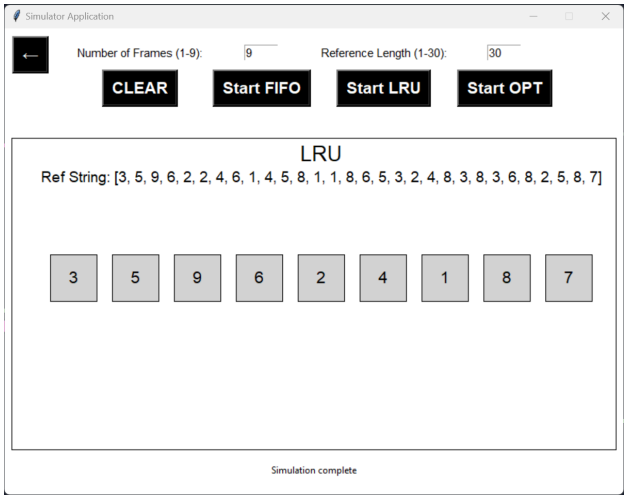


LRU Output

LRU selects for eviction the page whose last access is furthest in the past, thereby favoring retention of recently referenced data. For each of the three configurations mentioned under FIFO (3 frames with string length 10, 6 frames with length 20, and 9 frames with length 20), we captured every frame update from start to finish. These images illustrate how hits move pages to the most-recent end of the recency list and misses replace the least recently used frame according to the same test vectors.

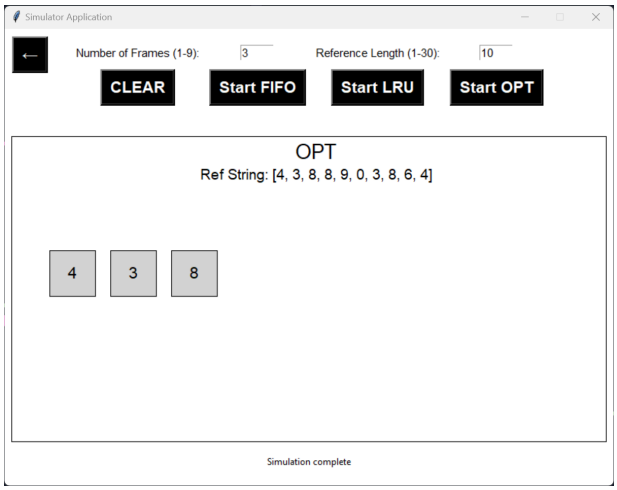


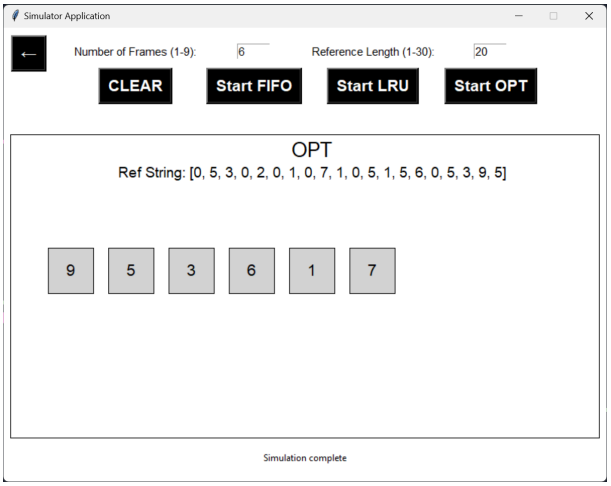


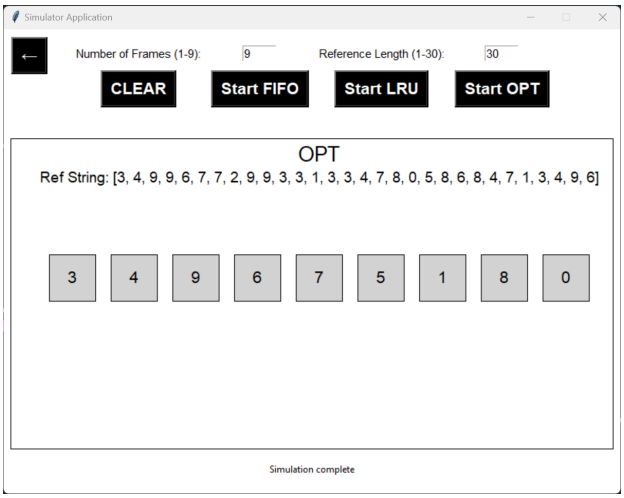


OPT Output

The OPT algorithm evicts the page whose next use lies farthest in the future, achieving the theoretical minimum page faults when the entire reference trace is known. Using the identical three scenarios outlined for FIFO (3 frames and 10-element reference string, 6 frames with 20 references, and 9 frames with 20 references), each screen capture details every decision point from the empty initialization through each replacement to the final frame state. This consistent setup highlights how OPT's clairvoyant strategy compares under the same workloads







Conclusion

The Optimal algorithm always produced the minimum number of page faults by replacing the page whose next reference is furthest in the future. Although this ensures the lowest fault rate in case the complete reference sequence is known, it involves checking all future requests prior to each replacement—a quantity of work that would cause a real operating system to slow dramatically.

The Least Recently Used algorithm closely tracked Optimal's fault performance while requiring much less effort. It maintains a simple list of pages sorted by the order they were last accessed and always discards the one at the rear. This results in every decision being fast and predictable, with near-optimal performance without the overhead of looking ahead.

First-In First-Out was the easiest to construct and visualize, based on a simple queue: pages arrive at the end and depart from the front. While it operated with little bookkeeping, its error rates were greater and occasionally even rose with added frames—a peculiarity which can produce counterintuitive consequences. Its simplicity does provide an easy reference point for judging more complex algorithms.

In all three algorithms, increasing the number of frames decreased faults but with diminishing returns. Optimal took the shortest number of frames to achieve its minimum number of faults, LRU took slightly more, and FIFO had the most irregularly spaced improvements. With more time and resources, adding cache-level simulation, evaluating adaptive eviction strategies such as CLOCK or ARC, and constructing real-time fault tracking dashboards would provide greater insight and assist in optimizing system parameters.