**CASE STUDY**

**Page Replacement Algorithm**

**Simulator**

**\**

**Prepared by:**

**Espinosa, Lazhar Angelo**

**Submitted To:**

**Jo Anne Cura**

**Instructor**

Table of Contents

[**Abstract** 3](#_Toc198580842)

[**1. Introduction** 3](#_Toc198580843)

[**2. Methodology** 4](#_Toc198580844)

[2.1 Algorithm Design 4](#_Toc198580845)

[2.2 Reference String Generation 4](#_Toc198580846)

[2.3 GUI Implementation 4](#_Toc198580847)

[**3. Implementation** 5](#_Toc198580848)

[3.1 Tools and Libraries 5](#_Toc198580849)

[3.2 Code Structure 5](#_Toc198580850)

[3.3 Key Features 5](#_Toc198580851)

[**4. Results and Discussion** 6](#_Toc198580853)

[**4.1 Experimental Setup** 6](#_Toc198580854)

[4.2 Observations 6](#_Toc198580855)

[**5. Conclusion** 7](#_Toc198580856)

## **Abstract**

This paper presents a simulation of three-page replacement algorithms—First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal (OPT)—to analyze their performance in managing page faults under varying memory constraints. A graphical user interface (GUI) built with Python’s Tkinter library enables users to input parameters, generate random page-reference strings, and visualize results. The study demonstrates that the Optimal algorithm consistently outperforms FIFO and LRU in minimizing page faults, though its reliance on future knowledge limits practical applicability. The GUI provides an intuitive platform for comparing algorithmic efficiency, making it a valuable educational tool for understanding virtual memory management.

Keywords: Page replacement algorithms, FIFO, LRU, OPT, virtual memory management, Tkinter GUI.

## **1. Introduction**

Page replacement algorithms are critical in operating systems to manage memory efficiently when the number of page frames is limited. This project implements and evaluates three widely studied algorithms:

1. **FIFO**: Replaces the oldest page in memory.
2. **LRU**: Replaces the least recently used page.
3. **OPT**: Replaces the page not used for the longest future time (theoretical optimal).

A GUI is integrated to facilitate user interaction, enabling dynamic testing of these algorithms under configurable parameters. The simulation highlights the trade-offs between implementation complexity and fault reduction.

## **2. Methodology**

### **2.1 Algorithm Design**

1. FIFO: Uses a queue structure. The oldest page is evicted on a page fault.
2. LRU: Maintains a list ordered by recency. The least recently accessed page is replaced.
3. OPT: Requires future knowledge of the reference string. The page with the farthest next use is evicted.

### **2.2 Reference String Generation**

A random sequence of integers (0–9) is generated to simulate page requests. Users can specify the length or input a custom string.

### **2.3 GUI Implementation**

The Tkinter-based GUI includes:

* Input fields for the number of frames and reference string length.
* A text box for custom reference strings.
* Buttons to generate strings and execute algorithms.
* A scrollable results panel displaying page faults.

## **3. Implementation**

### **3.1 Tools and Libraries**

* Python 3.x: Core programming language.
* Tkinter: GUI development.
* Random Module: Generates reference strings.

### **3.2 Code Structure**

1. Core Functions:
   * fifo(), lru(), opt() simulate each algorithm.
   * generate\_reference\_string() creates random page requests.
2. GUI Class:
   * PageReplacementApp initializes the interface and handles user interactions.

### **3.3 Key Features**

* Dynamic Inputs: Users can specify frames (1–10) and reference string length (≥1).
* Custom Reference Strings: Optional manual input for targeted testing.
* Real-Time Results: Immediate display of page faults for all algorithms.

## 

## **4. Results and Discussion**

## **4.1 Experimental Setup**

* Test Case 1:
  + Input: 3 frames, reference string length = 10.
  + Output: FIFO (6 faults), LRU (5 faults), OPT (5 faults).
* Test Case 2:
  + Input: 5 frames, reference string length = 20.
  + Output: FIFO (9 faults), LRU (8 faults), OPT (7 faults).

### **4.2 Observations**

1. FIFO suffers from Belady’s anomaly in some cases.
2. LRU performs better than FIFO by leveraging recency but requires overhead to track usage.
3. OPT serves as a theoretical benchmark but is unimplementable in real systems due to its reliance on future knowledge.

## **5. Conclusion**

This project successfully simulates FIFO, LRU, and OPT algorithms, demonstrating their relative efficiencies in minimizing page faults. The GUI enhances accessibility, enabling users to experiment with diverse scenarios. Future work could extend the tool to simulate other algorithms (e.g., Clock) or integrate real-time memory usage visualization.