Assignment 5 Report
Name: Miguelangel Tamargo
Panther ID: 5866999

In this assignment, I developed a C program that simulates the First-In-First-Out (FIFO) and Least Recently Used (LRU) page replacement algorithms. The program reads a sequence of page references from a specified file and determines the number of page faults that occur under each policy given a fixed number of memory frames. The primary objective was to understand and implement these fundamental memory management strategies, ensuring efficient handling of page faults and optimal utilization of memory frames.

The design choices focused on creating a modular and maintainable codebase. I structured the program into distinct functions for validating command-line arguments, reading page references from a file, and implementing each page replacement strategy. This separation of concerns not only enhanced code readability but also facilitated easier debugging and potential future extensions. Dynamic memory allocation was employed to handle an arbitrary number of page references, ensuring the program's flexibility regardless of input size. Sentinel values (-1) were used to indicate empty frames, simplifying the logic for detecting available memory slots and managing page hits or faults. Additionally, for the LRU implementation, an auxiliary counter array tracked the usage of each page, enabling efficient identification of the least recently used page without incurring significant computational overhead.

One of the main challenges encountered was implementing the LRU algorithm efficiently. Tracking the usage of each page required an effective mechanism to update and compare usage times without introducing excessive complexity or performance penalties. I addressed this by using a separate lru_counter array that recorded the index of the last occurrence of each page. This approach allowed quick identification of the least recently used page by simply finding the minimum value in the counter array. Another challenge was managing dynamic memory allocation, especially ensuring that the program could handle large sequences of page references without memory leaks or corruption. I implemented a doubling strategy with realloc to resize the page references array as needed, coupled with thorough error checking after each memory operation to handle allocation failures gracefully.

Handling edge cases also posed a significant challenge. Ensuring that the program correctly managed scenarios such as initially empty frames, all frames being occupied, and pages being referenced multiple times in rapid succession required careful logic implementation. I utilized sentinel values to denote empty frames and implemented flags to detect page hits, ensuring accurate recording and management of page faults.

This assignment was both challenging and enlightening. Implementing the FIFO and LRU algorithms deepened my understanding of memory management principles within operating systems. Working with dynamic memory allocation in C reinforced the importance of meticulous memory management practices to prevent leaks and ensure program stability. The process of translating theoretical algorithms into practical code highlighted the intricacies involved in algorithmic implementation and the trade-offs between simplicity and performance. Overcoming challenges related to efficient tracking mechanisms and dynamic array resizing enhanced my problem-solving skills and boosted my confidence in handling complex programming tasks.

To conclude, this project not only strengthened my programming skills in C but also provided valuable insights into the operational mechanics of page replacement algorithms. The experience underscored the significance of modular design, robust error handling, and efficient memory management in developing reliable and scalable software. Successfully implementing

FIFO and LRU page replacement strategies has prepared me for tackling more advanced concepts in operating systems and low-level programming, fostering a deeper appreciation for the sophisticated processes that underpin modern computing systems.