Page Replacement Algorithms – FIFO and LRU

Anthony Carvalho - 6365160

Thales Moraes - 6332398

This project implements two-page replacement algorithms: FIFO (First-In, First-Out) and LRU (Least Recently Used), simulating how operating systems manage memory pages. The program calculates page faults and outputs the final state of memory for each replacement policy.

Objective

The goal of this project is to understand and simulate page replacement mechanisms by reading a page reference string from the command line, simulating FIFO and LRU replacement policies, and calculating and displaying page faults and the final memory state for both algorithms.

Program Details

The program accepts two inputs: the number of frames (ranging from 1 to 10) and a page reference string separated by spaces. For each algorithm, it calculates the total number of page faults and displays the final memory state.

The sample command line input is:
./main 3 7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0 1

@output:

acarv048@ocelot:~/COP4610/PA5 133% ./main 3 7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0 1 7 0 1

FIFO: 15 page faults

Final state of memory: 7 0 1

LRU: 12 page faults

Final state of memory: 1 0 7

acarv048@ocelot:~/COP4610/PA5 134%

Screenshot of Output

(Attached is the screenshot of the output for this section.)

First-In, First-Out

The fifo function implements the First-In, First-Out page replacement algorithm. It replaces the oldest page in a fixed number of frames whenever a new page is needed and no free frame is available. The function counts page faults, updates the frames in a circular order, and outputs the total faults and the final memory state, providing a clear view of FIFO's performance.

```c
// Function to implement FIFO page replacement
void fifo(int *pages, int num_pages, int num_frames) {
    int *frame = calloc(num_frames, sizeof(int));
    int faults = 0, next_to_replace = 0;

    // Initialize frames as empty
    for (int i = 0; i < num_frames; i++) {
        frame[i] = -1;
    }

    for (int i = 0; i < num_pages; i++) {
        int page = pages[i];
        int found = 0;

        // Check if the page is already in the frame
        for (int j = 0; j < num_frames; j++) {
            if (frame[j] == page) {
                found = 1;
                break;
            }
        }

        // If page not found, replace using FIFO
        if (!found) {
            frame[next_to_replace] = page;
            next_to_replace = (next_to_replace + 1) % num_frames;
            faults++;
        }
    }

    // Output results
    printf("FIFO: %d page faults\n", faults);
    printf("Final state of memory: ");
    for (int i = 0; i < num_frames; i++) {
        printf("%d ", frame[i]);
    }
    printf("\n");
    free(frame);
}
```

LRU Page Replacment

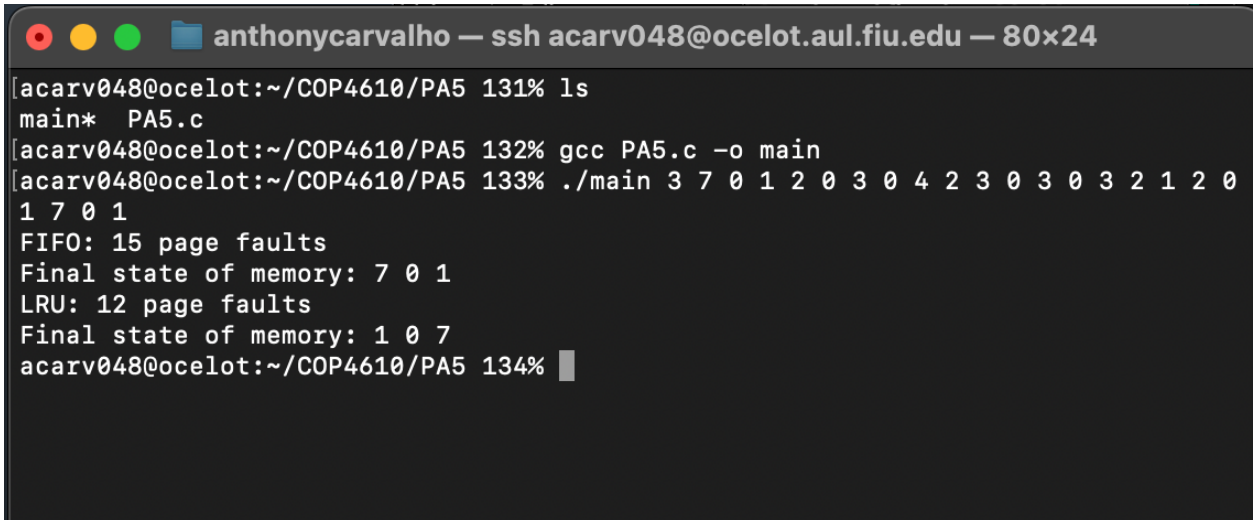
The lru function implements the Least Recently Used (LRU) page replacement algorithm. It tracks how recently each page in memory was accessed using an auxiliary array, last_used. When a page is not found in memory, it identifies and replaces the least recently used page. This process ensures that frequently accessed pages remain in memory. The function calculates the total page faults and outputs the final memory state, demonstrating the efficiency of LRU in minimizing page replacement operations.

```c
// Function implement LRU page replacement
void lru(int *pages, int num_pages, int num_frames) {
    int *frame = calloc(num_frames, sizeof(int));
    int *last_used = calloc(num_frames, sizeof(int));
    int faults = 0;
    // initialize frames as empty
    for (int i = 0; i < num_frames; i++) {
        frame[i] = -1;
        last_used[i] = -1;
    }
    for (int i = 0; i < num_pages; i++) {
        int page = pages[i];
        int found = 0;
        // Check if the page is already in the frame
        for (int j = 0; j < num_frames; j++) {
            if (frame[j] == page) {
                found = 1;
                last_used[j] = i; // update the last used time
                break;
            }
        }
        // if the page not found, replace using LRU
        if (!found) {
            int lru_index = 0;

            // Find the least recently used page
            for (int j = 1; j < num_frames; j++) {
                if (last_used[j] < last_used[lru_index]) {
                    lru_index = j;
                }
            }

            frame[lru_index] = page;
            last_used[lru_index] = i;
            faults++;
        }
    }
    // Output results
    printf("LRU: %d page faults\n", faults);
    printf("Final state of memory: ");
    for (int i = 0; i < num_frames; i++) {
        printf("%d ", frame[i]);
```

4, Column 13


anthonycarvalho — ssh acarv048@ocelot.aul.fiu.edu — 80×24
[acarv048@ocelot:~/COP4610/PA5 131% ls
main* PA5.c
[acarv048@ocelot:~/COP4610/PA5 132% gcc PA5.c -o main
[acarv048@ocelot:~/COP4610/PA5 133% ./main 3 7 0 1 2 0 3 0 4 2 3 0 3 0 3 2 1 2 0
1 7 0 1
FIFO: 15 page faults
Final state of memory: 7 0 1
LRU: 12 page faults
Final state of memory: 1 0 7
acarv048@ocelot:~/COP4610/PA5 134%