LAB # 7

Name: Aliyan Ahmed Cheema

Reg # FA22-BCE-028

PAGE REPLACEMENT STRATEGIES

// Task 1 (a)

// add the code in it to find the hit ratio and miss ratio

// execute the code for different string of same length say if you have run before with length 10

// and frame size 3

// find the hit ratio and miss ratio

// Task 1(b)

// you need to take length atleast 20 in the previous two scenarios because

// in the next task we are increasing the frame size so length 10 will not give us better results

// Task 1(c) Increase the frame size to 4 and observe

// the output with hit ratio and miss ratio what results you observed with this increase of frame size

// execute Task 1(c) i.e increase frame size of 4 with the same last two strings in the Task 1(a) and Task 1(b)

#include <stdio.h>

int main()

{

    // Declare variables for page replacement simulation

    int i, j, n, a[50], frame[10], no, k, avail, count = 0, hits = 0;

    // Prompt the user to enter the total number of pages

    printf("\n ENTER THE NUMBER OF PAGES:\n");

    scanf("%d", &n); // Read the number of pages

    // Prompt the user to enter the sequence of page numbers

    printf("\n ENTER THE PAGE NUMBER :\n");

    for (i = 1; i <= n; i++) // Loop to input each page number

        scanf("%d", &a[i]); // Store page numbers in array `a`

    // Prompt the user to input the number of frames available

    printf("\n ENTER THE NUMBER OF FRAMES :");

    scanf("%d", &no); // Read the number of frames

    // Initialize all frame slots to -1 to indicate they are empty

    for (i = 0; i < no; i++)

        frame[i] = -1;

    j = 0; // Initialize pointer for FIFO replacement policy

    printf("\tref string\t page frames\n"); // Header for output table

    // Simulate the page replacement process

    for (i = 1; i <= n; i++){

        printf("%d\t\t", a[i]); // Display the current page being accessed

        avail = 0; // Reset availability flag for each page check

        // Check if the page is already present in any frame

        for (k = 0; k < no; k++){

            if (frame[k] == a[i]){ // Page hit

                avail = 1; // Set availability flag to indicate a hit

                break; // Exit the loop as the page is found

            }

        }

        // Handle a page fault if the page is not found in any frame

        if (avail == 0)

        {

            frame[j] = a[i];  // Replace the page in the current frame

            j = (j + 1) % no; // Move pointer to the next frame in a circular manner

            count++;          // Increment page fault counter

            // Display the current state of the frames

            for (k = 0; k < no; k++)

                printf("%d\t", frame[k]);

        }

        else{

            hits++; // Increment the hit counter when the page is already in a frame

        }

        printf("\n"); // Print a newline after each step for clarity

    }

    // Display the results: total page faults and hits

    printf("Page Faults(Page Miss): %d\n", count);

    printf("Hits: %d\n", hits);

    // Calculate and display the hit ratio and miss ratio

    float hit\_ratio = (float)hits / n;   // Calculate hit ratio

    float miss\_ratio = (float)count / n; // Calculate miss ratio

    printf("Hit Ratio: %.2f\n", hit\_ratio);                      // Display hit ratio

    printf("Miss Ratio: %.2f\n", miss\_ratio);                    // Display miss ratio

    printf("\nHit Ratio Percentage: %.2f%%\n", hit\_ratio \* 100); // Display hit ratio in percentage

    printf("Miss Ratio Percentage: %.2f%%\n", miss\_ratio \* 100); // Display miss ratio in percentage

    return 0; // Return 0 to indicate successful execution of the program

}

// LRU Page Replacement Algorithm

/\*

ALGO :

1. Start the process

2. Declare the size

3. Get the number of pages to be inserted

4. Get the value

5. Declare counter and stack

6. Select the least recently used page by counter value

7. Stack them according the selection.

8.  Display the values

9. Stop the process

 PROGRAM : \*/

 #include <stdio.h> // Include standard input-output library

int main(){ // Main function: Entry point of the program

    int q[20], p[50];          // `q` holds the frames, `p` holds the page reference string

    int c = 0, c1, d, f;       // `c` is the page fault counter, `c1` checks for page hits

    int i, j, k = 0, n, r, t;  // Loop variables, `k` tracks frames filled

    int b[20], c2[20];         // `b` and `c2` used for LRU calculations

    // Prompt user for the number of pages in the reference string

    printf("Enter number of pages: ");

    scanf("%d", &n); // Input total number of pages

    // Prompt user to enter the reference string

    printf("Enter the reference string: ");

    for (i = 0; i < n; i++)

        scanf("%d", &p[i]); // Input page numbers into the array `p`

    // Prompt user to input the number of frames

    printf("Enter number of frames: ");

    scanf("%d", &f); // Input frame size

    // Initialize the first page in the first frame

    q[k] = p[k]; // Place the first page in the first frame

    printf("\n\t%d\n", q[k]); // Display the current frame content

    c++; // Increment the page fault count

    k++; // Increment the frame counter

    // Process the remaining pages in the reference string

    for (i = 1; i < n; i++){

        c1 = 0; // Reset the hit counter for each page

        // Check if the page is already present in the frames

        for (j = 0; j < f; j++) {

            if (p[i] != q[j]) // If the page is not found in the frame

                c1++; // Increment the counter

        }

        // If the page is not in any frame (a page fault occurs)

        if (c1 == f) {

            c++; // Increment the page fault counter

            // If there is space available in the frames

            if (k < f){

                q[k] = p[i]; // Add the page to the next available frame

                k++; // Increment the frame counter

                for (j = 0; j < k; j++)

                    printf("\t%d", q[j]); // Display current frame content

                printf("\n");

            }

            Else{ // If frames are full, use LRU replacement policy

                for (r = 0; r < f; r++) // For each frame

                {

                    c2[r] = 0; // Initialize counter for LRU calculation

                    for (j = i - 1; j >= 0; j--) // Check previous references

                    {

                        if (q[r] != p[j])

                            c2[r]++; // Increment the counter if page not matched

                        else

                            break; // Stop when the page is found

                    }

                }

                // Copy LRU counters to another array for sorting

                for (r = 0; r < f; r++)

                    b[r] = c2[r];

                // Sort LRU counters in descending order

                for (r = 0; r < f; r++){

                    for (j = r + 1; j < f; j++) {

                        if (b[r] < b[j]) // Swap if out of order

                        {

                            t = b[r];

                            b[r] = b[j];

                            b[j] = t;

                        }

                    }

                }

                // Replace the least recently used page

                for (r = 0; r < f; r++) {

                    if (c2[r] == b[0]) // Find the page with the highest counter

                        q[r] = p[i]; // Replace it with the current page

                    printf("\t%d", q[r]); // Display current frame content

                }

                printf("\n");

            }

        }

    }

    // Display the total number of page faults

    printf("\nThe number of page faults is %d", c);

    // Calculate and display hit and miss ratios

    int hits = n - c; // Hits are total accesses minus page faults

    float hit\_ratio = (float)hits / n;   // Calculate hit ratio

    float miss\_ratio = (float)c / n;     // Calculate miss ratio

    printf("\nHits: %d\n", hits);                       // Display total hits

    printf("Hit Ratio: %.2f\n", hit\_ratio);             // Display hit ratio

    printf("Miss Ratio: %.2f\n", miss\_ratio);           // Display miss ratio

    printf("Hit Ratio Percentage: %.2f%%\n", hit\_ratio \* 100); // Display hit ratio percentage

    printf("Miss Ratio Percentage: %.2f%%\n", miss\_ratio \* 100); // Display miss ratio percentage

    return 0; // Return 0 to indicate successful execution

}

/\*

Task 3a. Comment the code

Task 3b. Compile and Run the code with same 2 page reference strings in the previous tasks and display the hit ratio and miss ratio

set Frame size = 3 in Task 3b

Check that whether the hit ratio improved in the optimal strategy

Task 3c. Run the code with frame size=4 and compare the performance with itself at frame=3 and also with FIFO and LRU at frame =4

Performance means hit ratio and miss ratio

Program :

\*/

#include <stdio.h> // Include standard input-output library

int main() // Main function: Entry point of the program

{

    // Variable declarations

    int no\_of\_frames, no\_of\_pages;         // Number of frames and pages

    int frames[10], pages[30], temp[10];  // Arrays for frames, pages, and a temporary array for calculations

    int flag1, flag2, flag3;              // Flags to track conditions (page hit, empty frame, etc.)

    int i, j, k, pos, max;                // Loop variables and helpers

    int faults = 0;                       // Counter for page faults

    // Input the number of frames

    printf("Enter number of frames: ");

    scanf("%d", &no\_of\_frames);

    // Input the number of pages

    printf("Enter number of pages: ");

    scanf("%d", &no\_of\_pages);

    // Input the page reference string

    printf("Enter page reference string: ");

    for (i = 0; i < no\_of\_pages; ++i) {

        scanf("%d", &pages[i]); // Read each page number into the array `pages`

    }

    // Initialize all frames to -1 (indicating empty frames)

    for (i = 0; i < no\_of\_frames; ++i) {

        frames[i] = -1;

    }

    // Process each page in the reference string

    for (i = 0; i < no\_of\_pages; ++i) {

        flag1 = flag2 = 0; // Reset flags for the current page

        // Check if the page is already present in the frames (page hit)

        for (j = 0; j < no\_of\_frames; ++j) {

            if (frames[j] == pages[i]) {

                flag1 = flag2 = 1; // Set flags if page is found (no page fault)

                break;

            }

        }

        // If the page is not in the frames

        if (flag1 == 0) {

            // Look for an empty frame to place the new page

            for (j = 0; j < no\_of\_frames; ++j) {

                if (frames[j] == -1){ // If an empty frame is found

                    faults++;         // Increment the page fault counter

                    frames[j] = pages[i]; // Place the page in the empty frame

                    flag2 = 1;        // Mark the page as placed

                    break;

                }

            }

        }

        // If no empty frame is found, use the Optimal Page Replacement strategy

        if (flag2 == 0) {

            flag3 = 0; // Reset flag for replacement calculation

            // Calculate the future use of each frame

            for (j = 0; j < no\_of\_frames; ++j) {

                temp[j] = -1; // Initialize future use to -1 (not found)

                // Check how far in the future each page in the frames is referenced

                for (k = i + 1; k < no\_of\_pages; ++k) {

                    if (frames[j] == pages[k]) {// If the page is found in the future

                        temp[j] = k; // Record the future index of the page

                        break;

                    }

                }

            }

            // Find a frame that is not used in the future

            for (j = 0; j < no\_of\_frames; ++j) {

                if (temp[j] == -1){ // If a frame's page is not referenced again

                    pos = j;       // Select that frame for replacement

                    flag3 = 1;     // Mark the frame for replacement

                    break;

                }

            }

            // If all frames' pages are referenced in the future, replace the page used farthest in the future

            if (flag3 == 0) {

                max = temp[0]; // Start with the first frame's future use

                pos = 0;       // Assume the first frame will be replaced

                for (j = 1; j < no\_of\_frames; ++j) {

                    if (temp[j] > max) {// If another page is used later than the current maximum

                        max = temp[j]; // Update the maximum future use

                        pos = j;       // Update the frame position to replace

                    }

                }

            }

            // Replace the page in the selected frame

            frames[pos] = pages[i];

            faults++; // Increment the page fault counter

        }

        // Display the current state of frames

        printf("\n");

        for (j = 0; j < no\_of\_frames; ++j) {

            printf("%d\t", frames[j]);

        }

    }

    // Display the total number of page faults

    printf("\n\nTotal Page Faults = %d", faults);

    // Calculate and display hit and miss ratios

    int hits = no\_of\_pages - faults; // Calculate hits as total accesses minus faults

    float hit\_ratio = (float)hits / no\_of\_pages; // Calculate hit ratio

    float miss\_ratio = (float)faults / no\_of\_pages; // Calculate miss ratio

    printf("\nHits: %d\n", hits);                       // Display total hits

    printf("Hit Ratio: %.2f\n", hit\_ratio);             // Display hit ratio

    printf("Miss Ratio: %.2f\n", miss\_ratio);           // Display miss ratio

    printf("Hit Ratio Percentage: %.2f%%\n", hit\_ratio \* 100); // Display hit ratio percentage

    printf("Miss Ratio Percentage: %.2f%%\n", miss\_ratio \* 100); // Display miss ratio percentage

    return 0; // Indicate successful program execution

}