Page replacement algorithm

#include <stdio.h>

#define MAX\_FRAMES 3 // Number of frames in memory

// FIFO Algorithm

void fifo(int pages[], int n) {

int frames[MAX\_FRAMES];

int page\_faults = 0;

int index = 0;

// Initialize all frames to -1 (empty)

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1;

}

for (int i = 0; i < n; i++) {

int found = 0;

// Check if the page is already in one of the frames

for (int j = 0; j < MAX\_FRAMES; j++) {

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

found = 1;

break;

}

}

// If page is not found, replace the oldest page (FIFO)

if (!found) {

frames[index] = pages[i];

index = (index + 1) % MAX\_FRAMES; // Cycle the frame index

page\_faults++;

}

// Print the current state of the frames

printf("Page: %d Frames: ", pages[i]);

for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] != -1)

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

}

printf("\n");

}

printf("Total page faults (FIFO): %d\n", page\_faults);

}

// Optimal Algorithm

int findOptimal(int frames[], int n, int pages[], int current\_index) {

int farthest = -1;

int index\_to\_replace = -1;

// Check each frame to see which page will not be used in the future or used farthest away

for (int j = 0; j < MAX\_FRAMES; j++) {

int found = 0;

// Look ahead in the future reference string to find when the page will be used again

for (int k = current\_index; k < n; k++) {

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

if (k > farthest) {

farthest = k;

index\_to\_replace = j;

}

found = 1;

break;

}

}

// If the page is not found in the future, return this index to replace

if (!found) {

return j;

}

}

return index\_to\_replace;

}

void optimal(int pages[], int n) {

int frames[MAX\_FRAMES];

int page\_faults = 0;

// Initialize all frames to -1 (empty)

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1;

}

for (int i = 0; i < n; i++) {

int found = 0;

// Check if the page is already in one of the frames

for (int j = 0; j < MAX\_FRAMES; j++) {

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

found = 1;

break;

}

}

// If page is not found, replace the optimal page

if (!found) {

int replaceIndex = findOptimal(frames, n, pages, i + 1);

frames[replaceIndex] = pages[i];

page\_faults++;

}

// Print the current state of the frames

printf("Page: %d Frames: ", pages[i]);

for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] != -1)

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

}

printf("\n");

}

printf("Total page faults (Optimal): %d\n", page\_faults);

}

// LRU Algorithm

void lru(int pages[], int n) {

int frames[MAX\_FRAMES];

int page\_faults = 0;

int access\_time[MAX\_FRAMES]; // Track the last access time for LRU

// Initialize all frames to -1 (empty)

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1;

access\_time[i] = -1; // Initialize access times to -1

}

for (int i = 0; i < n; i++) {

int found = 0;

// Check if the page is already in one of the frames

for (int j = 0; j < MAX\_FRAMES; j++) {

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

found = 1;

access\_time[j] = i; // Update the access time

break;

}

}

// If page is not found, replace the least recently used page

if (!found) {

int lru\_index = 0;

for (int j = 1; j < MAX\_FRAMES; j++) {

if (access\_time[j] < access\_time[lru\_index]) {

lru\_index = j; // Find the least recently used page

}

}

frames[lru\_index] = pages[i];

access\_time[lru\_index] = i; // Update the access time for the new page

page\_faults++;

}

// Print the current state of the frames

printf("Page: %d Frames: ", pages[i]);

for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] != -1)

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

}

printf("\n");

}

printf("Total page faults (LRU): %d\n", page\_faults);

}

int main() {

int pages[] ={1,4,2,3,2,1

}; // Page reference string

int n = sizeof(pages) / sizeof(pages[0]);

int choice;

printf("Choose page replacement algorithm:\n");

printf("1. FIFO\n");

printf("2. LRU\n");

printf("3. Optimal\n");

printf("Enter your choice (1/2/3): ");

scanf("%d", &choice);

switch(choice) {

case 1:

printf("FIFO Page Replacement:\n");

fifo(pages, n);

break;

case 2:

printf("LRU Page Replacement:\n");

lru(pages, n);

break;

case 3:

printf("Optimal Page Replacement:\n");

optimal(pages, n);

break;

default:

printf("Invalid choice! Please enter 1, 2, or 3.\n");

}

return 0;

}

Output:

