**Name : Tanishq Thuse**

**Branch : CS(AI)**

**Year : SY**

**Div : B**

**Roll no. : 60**

**PRN : 12310237**

**Title : OS Assignment-7 Implement page replacement algorithms**

**Q1) FIFO**

**Code :**

#include <stdio.h>

#include <stdbool.h>

void display(int frames[], int frameSize) {

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

if (frames[i] == -1)

printf("- ");

else

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

}

printf("\n");

}

// FIFO Page Replacement

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

int frames[frameSize], pageFaults = 0, next = 0;

// Initialize frames to empty

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

frames[i] = -1;

}

printf("\nFIFO Page Replacement\n");

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

int page = pages[i];

bool found = false;

// Check if page is already in a frame

for (int j = 0; j < frameSize; j++) {

if (frames[j] == page) {

found = true;

break;

}

}

if (!found) {

// Replace page using FIFO

frames[next] = page;

next = (next + 1) % frameSize;

pageFaults++;

}

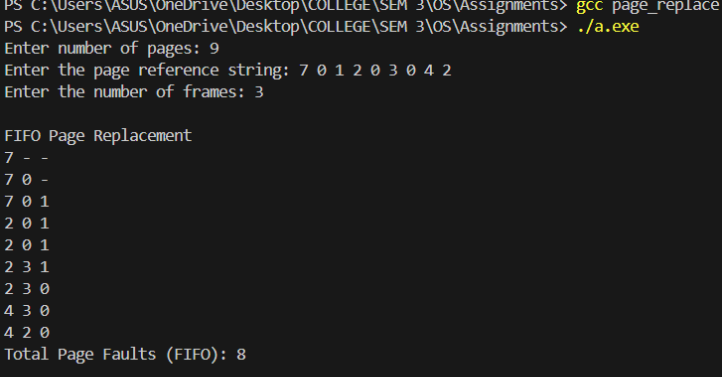
display(frames, frameSize);

}

printf("Total Page Faults (FIFO): %d\n", pageFaults);

}

**Output :**



**Q2) Optimal Page Replacement**

**Code :**

#include <stdio.h>

#include <stdbool.h>

// Function to display the current state of the frames

void display(int frames[], int frameSize) {

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

if (frames[i] == -1)

printf("- ");

else

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

}

printf("\n");

}

// Optimal Page Replacement

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

int frames[frameSize], pageFaults = 0;

// Initialize all frames to empty (-1)

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

frames[i] = -1;

}

printf("\nOptimal Page Replacement\n");

// Process each page one by one

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

int page = pages[i];

bool found = false;

// Check if the page is already in a frame

for (int j = 0; j < frameSize; j++) {

if (frames[j] == page) {

found = true;

break;

}

}

if (!found) {

// Find the page to replace

int farthest = -1, index = -1;

for (int j = 0; j < frameSize; j++) {

if (frames[j] == -1) {

index = j; // Empty frame found

break;

}

int k;

// Look ahead to find the next occurrence of the page

for (k = i + 1; k < n; k++) {

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

if (k > farthest) {

farthest = k;

index = j;

}

break;

}

}

// If the page is not referenced again, select it for replacement

if (k == n) {

index = j;

break;

}

}

// Replace the selected page with the new page

frames[index] = page;

pageFaults++;

}

display(frames, frameSize);

}

// Print total page faults

printf("Total Page Faults (Optimal): %d\n", pageFaults);

}

int main() {

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

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

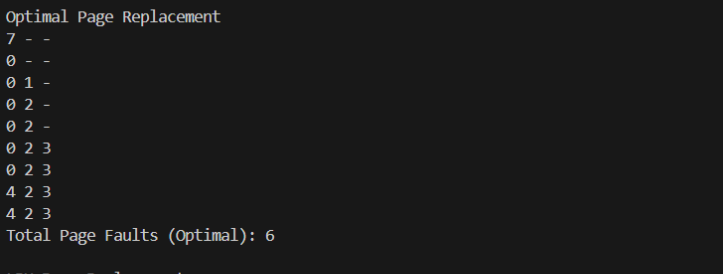
int frameSize = 3;

optimal(pages, n, frameSize);

return 0;

}

**Output :**



**Q3) LRU Page Replacement**

**Code :**

#include <stdio.h>

#include <stdbool.h>

// Function to display the current state of the frames

void display(int frames[], int frameSize) {

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

if (frames[i] == -1)

printf("- ");

else

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

}

printf("\n");

}

// LRU Page Replacement Algorithm

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

int frames[frameSize], pageFaults = 0, counter[frameSize];

// Initialize all frames to -1 (empty) and counters to 0

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

frames[i] = -1;

counter[i] = 0;

}

printf("\nLRU Page Replacement\n");

// Process each page one by one

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

int page = pages[i];

bool found = false;

// Check if the page is already in a frame

for (int j = 0; j < frameSize; j++) {

if (frames[j] == page) {

found = true;

counter[j] = i; // Update the last time this page was used

break;

}

}

if (!found) {

// Find the least recently used page (based on the counter)

int lruIndex = 0;

for (int j = 1; j < frameSize; j++) {

if (counter[j] < counter[lruIndex]) {

lruIndex = j;

}

}

// Replace the least recently used page with the current page

frames[lruIndex] = page;

counter[lruIndex] = i;

pageFaults++;

}

display(frames, frameSize);

}

// Print total page faults

printf("Total Page Faults (LRU): %d\n", pageFaults);

}

int main() {

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

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

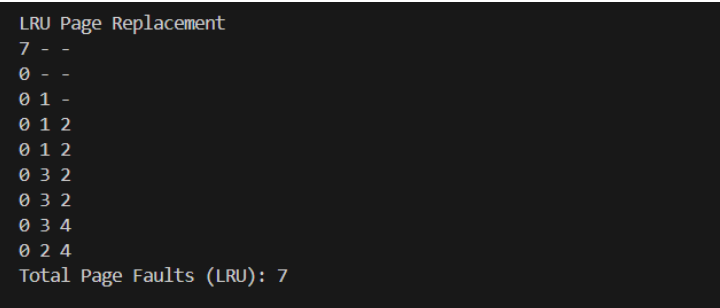
int frameSize = 3;

lru(pages, n, frameSize);

return 0;

}

Output:



Q4) Clock Page Replacement

**Code :**

#include <stdio.h>

#include <stdbool.h>

// Function to display the current state of the frames

void display(int frames[], int frameSize) {

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

if (frames[i] == -1)

printf("- ");

else

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

}

printf("\n");

}

// Clock Page Replacement Algorithm

void clock(int pages[], int n, int frameSize) {

int frames[frameSize], useBit[frameSize], pageFaults = 0, pointer = 0;

// Initialize frames to -1 (empty) and use bits to 0

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

frames[i] = -1;

useBit[i] = 0;

}

printf("\nClock Page Replacement\n");

// Process each page one by one

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

int page = pages[i];

bool found = false;

// Check if the page is already in a frame

for (int j = 0; j < frameSize; j++) {

if (frames[j] == page) {

useBit[j] = 1; // Set use bit to 1

found = true;

break;

}

}

if (!found) {

// Replace page using Clock algorithm

while (useBit[pointer] == 1) {

useBit[pointer] = 0; // Reset the use bit

pointer = (pointer + 1) % frameSize; // Move to the next frame

}

frames[pointer] = page; // Replace the page

useBit[pointer] = 1; // Set the use bit for the newly added page

pointer = (pointer + 1) % frameSize; // Move pointer to the next frame

pageFaults++;

}

display(frames, frameSize);

}

// Print total page faults

printf("Total Page Faults (Clock): %d\n", pageFaults);

}

int main() {

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

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

int frameSize = 3;

clock(pages, n, frameSize);

return 0;

}

**Output :**

