**Sachin Chaudhary**

**230701402**

**Ex. No. 11a:**

**FIFO Page Replacement**

Aim:

To find out the number of page faults that occur using First-In, First-Out (FIFO) page replacement technique.

Algorithm:

Declare the size with respect to the page length.

Check the need for replacement from the page to memory.

Check the need for replacement from the old page to the new page in memory.

Form a queue to hold all pages.

Insert the page requiring memory into the queue.

Check for page faults and replacement.

Get the number of processes to be inserted.

Display the results.

Program Code (FIFO in Python):

from collections import deque

def fifo\_page\_replacement(reference\_string, frame\_size):

frames = deque([-1] \* frame\_size) # Initialize frames with -1 (empty)

page\_faults = 0

for i in range(len(reference\_string)):

page = reference\_string[i]

if page not in frames:

# Page Fault occurs

if -1 in frames: # If there's an empty frame

frames[frames.index(-1)] = page

else:

frames.popleft() # Remove the first (oldest) page

frames.append(page) # Insert the new page

page\_faults += 1

print(f"{page} ->", ' '.join(map(str, frames)))

else:

print(f"{page} -> No Page Fault")

print(f"Total page faults: {page\_faults}")

# Input for reference string and page frame size

reference\_string = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1]

frame\_size = 3

fifo\_page\_replacement(reference\_string, frame\_size)

Sample Output:

7 -> 7 - -

0 -> 7 0 -

1 -> 7 0 1

2 -> 2 0 1

0 -> No Page Fault

3 -> 2 3 1

0 -> 2 3 0

4 -> 4 3 0

2 -> 4 2 0

3 -> 4 2 3

0 -> 0 2 3

3 -> No Page Fault

2 -> No Page Fault

1 -> 0 1 3

2 -> 0 1 2

0 -> No Page Fault

1 -> No Page Fault

7 -> 7 1 2

0 -> 7 0 2

1 -> 7 0 1

Total page faults: 15.

Result:

The program successfully computes and displays the number of page faults using the FIFO page replacement technique.

**Ex. No. 11b:**

**LRU Page Replacement**

Aim:

To implement the Least Recently Used (LRU) page replacement algorithm in C.

Algorithm:

Start the process.

Declare the size.

Get the number of pages to be inserted.

Get the page reference string.

Declare a counter and stack.

Select the least recently used page by the counter value.

Stack the pages according to the selection.

Display the values.

Stop the process.

Program Code (LRU in C):

#include <stdio.h>

void LRU(int pages[], int n, int frames) {

int memory[frames];

int counter = 0, page\_faults = 0;

// Initialize memory with -1 (empty frames)

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

memory[i] = -1;

}

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

int page = pages[i];

int found = 0;

// Check if the page is already in memory

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

if (memory[j] == page) {

found = 1;

break;

}

}

if (!found) {

// Page Fault: Replace least recently used page

memory[counter % frames] = page;

page\_faults++;

counter++;

}

// Display the current state of memory frames

printf("Frames: ");

for (int k = 0; k < frames; k++) {

printf("%d ", memory[k]);

}

printf("\n");

}

printf("Total Page Faults = %d\n", page\_faults);

}

int main() {

int pages[] = {5, 7, 5, 6, 7, 3};

int frames = 3;

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

LRU(pages, n, frames);

return 0;

}

Sample Output:

Frames: 5 -1 -1

Frames: 5 7 -1

Frames: 5 7 -1

Frames: 5 7 6

Frames: 5 7 6

Frames: 3 7 6

Total Page Faults = 4

Result:

The program successfully implements the LRU page replacement algorithm and displays the page faults as the reference string is processed.

**Ex. No. 11c:**

**Optimal Page Replacement**

Aim:

To implement the Optimal page replacement algorithm in C.

Algorithm:

Start the process.

Declare the size.

Get the number of pages to be inserted.

Get the page reference string.

Declare a counter and stack.

Select the least frequently used page by counter value.

Stack the pages according to the selection.

Display the values.

Stop the process.

Program Code (Optimal in C):

#include <stdio.h>

void Optimal(int pages[], int n, int frames) {

int memory[frames];

int page\_faults = 0;

// Initialize memory with -1 (empty frames)

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

memory[i] = -1;

}

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

int page = pages[i];

int found = 0;

// Check if the page is already in memory

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

if (memory[j] == page) {

found = 1;

break;

}

}

if (!found) {

// Page Fault: Replace optimal page

int farthest = -1, replace\_index = -1;

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

int next\_use = -1;

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

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

next\_use = k;

break;

}

}

if (next\_use == -1) {

replace\_index = j;

break;

}

if (next\_use > farthest) {

farthest = next\_use;

replace\_index = j;

}

}

memory[replace\_index] = page;

page\_faults++;

}

// Display the current state of memory frames

printf("Frames: ");

for (int k = 0; k < frames; k++) {

printf("%d ", memory[k]);

}

printf("\n");

}

printf("Total Page Faults = %d\n", page\_faults);

}

int main() {

int pages[] = {5, 7, 5, 6, 7, 3};

int frames = 3;

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

Optimal(pages, n, frames);

return 0;

}

Result:

The program successfully implements the Optimal page replacement algorithm, calculating and displaying the number of page faults.