# **Experiment No: 08**

## AIM: Implementation of various page replacement algorithms (FIFO, Optimal, LRU).

### **Objective:** To study and implement various Page Replacement Algorithms.

#### Theory:

**FIFO Page Replacement Algorithm:**

FIFO which is also called First In First Out is one of the types of Replacement Algorithms. This algorithm is used in a situation where an Operating system replaces an existing page with the help of memory by bringing a new page from the secondary memory. FIFO is the simplest among all algorithms which are responsible for maintaining all the pages in a queue for an operating system and also keeping track of all the pages in a queue. The older pages are kept in the front and the newer ones are kept at the end of the queue. Pages that are in the front are removed first and the pages which are demanded are added.

Algorithm:

1. Start traversing the pages.
2. Now declare the size w.r.t length of the Page.
3. Check need of the replacement from the page to memory.
4. Similarly, Check the need of the replacement from the old page to new page in memory.
5. Now form the queue to hold all pages.
6. Insert Require page memory into the queue.
7. Check bad replacements and page faults.
8. Get no of processes to be inserted.
9. Show the values
10. Stop

**Source code:**

#include<stdio.h>

int main()

{

int incomingStream[] = {4, 1, 2, 4, 5};

int pageFaults = 0;

int frames = 3;

int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");

int temp[frames];

for(m = 0; m < frames; m++)

{

temp[m] = -1;

}

for(m = 0; m < pages; m++)

{

s = 0;

for(n = 0; n < frames; n++)

{

if(incomingStream[m] == temp[n])

{

s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0))

{

temp[m] = incomingStream[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = incomingStream[m];

}

printf("\n");

printf("%d\t\t\t",incomingStream[m]);

for(n = 0; n < frames; n++)

{

if(temp[n] != -1)

printf(" %d\t\t\t", temp[n]);

else

printf(" - \t\t\t");

}

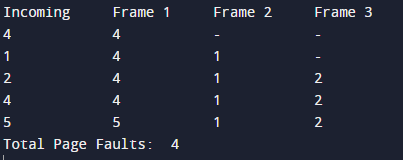
}

printf("\nTotal Page Faults:\t%d\n", pageFaults);

return 0;

}

**Output:**



#### LRU Page Replacement Algorithm:

Least Recently Used (LRU) page replacement algorithm works on the concept that the pages that are heavily used in previous instructions are likely to be used heavily in next instructions. And the page that are used very less are likely to be used less in future.

Whenever a page fault occurs, the page that is least recently used is removed from the memory frames. Page fault occurs when a referenced page in not found in the memory frames.

Algorithm:

Start the process

* 1. Declare the size
  2. Get the number of pages to be inserted
  3. Get the value
  4. Declare counter and stack
  5. Select the least recently used page by counter value
  6. Stack them according the selection.
  7. Display the values
  8. Stop the process

**Source Code:**

#include<stdio.h>

#include<limits.h>

int checkHit(int incomingPage, int queue[], int occupied){

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

if(incomingPage == queue[i])

return 1;

}

return 0;

}

void printFrame(int queue[], int occupied)

{

for(int i = 0; i < occupied; i++)

printf("%d\t\t\t",queue[i]);

}

int main()

{

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

// int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3};

int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};

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

int frames = 3;

int queue[n];

int distance[n];

int occupied = 0;

int pagefault = 0;

printf("Page\t Frame1 \t Frame2 \t Frame3\n");

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

{

printf("%d: \t\t",incomingStream[i]);

// what if currently in frame 7

// next item that appears also 7

// didnt write condition for HIT

if(checkHit(incomingStream[i], queue, occupied)){

printFrame(queue, occupied);

}

// filling when frame(s) is/are empty

else if(occupied < frames){

queue[occupied] = incomingStream[i];

pagefault++;

occupied++;

printFrame(queue, occupied);

}

else{

int max = INT\_MIN;

int index;

// get LRU distance for each item in frame

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

{

distance[j] = 0;

// traverse in reverse direction to find

// at what distance frame item occurred last

for(int k = i - 1; k >= 0; k--)

{

++distance[j];

if(queue[j] == incomingStream[k])

break;

}

// find frame item with max distance for LRU

// also notes the index of frame item in queue

// which appears furthest(max distance)

if(distance[j] > max){

max = distance[j];

index = j;

}

}

queue[index] = incomingStream[i];

printFrame(queue, occupied);

pagefault++;

}

printf("\n");

}

printf("Page Fault: %d",pagefault);

return 0;

}

**Output:**

