**Page Replacement Algorithms in OS**

On the paging page we saw the mapping of virtual memory to physical memory and how MMU does the paging process. But, what should happen when a new page comes is in scenerio. Which of the existing pages should be replaced and how to decide which one to replace.

On this page we will discuss all page replacement algorithms in operating system.

Page Replacement algorithms in operating system are different operational logics to decide which page should be replaced when a new page comes in the system.

**Types of Page Replacement Algorithms**

FIFO – First in First Out

LRU – Least Recently Used

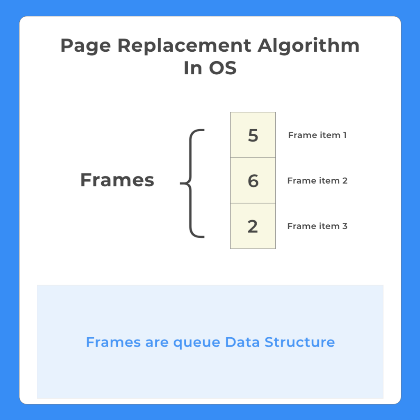
Optimal Page Replacement Algorithm

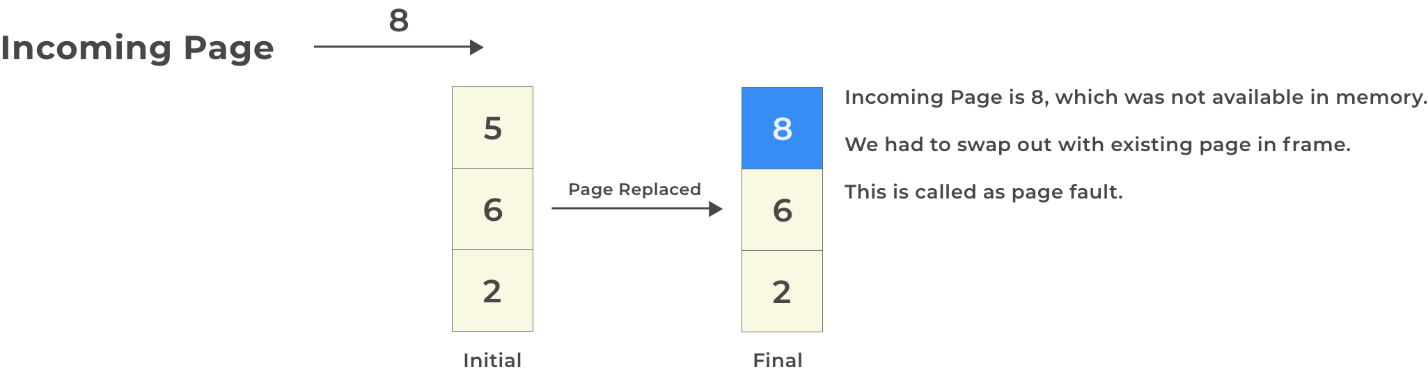
LFU – Least Frequently Used

MFU – Most Frequently Used

A Page Fault occurs when a program running on the CPU tries to access a page that is in the address space of that program, but the requested page is currently not loaded into the main physical memory, the RAM of the system

Page Fault – In simple terms, if an incoming stream item (page) is not available in any of the frames. Then page fault occurs.





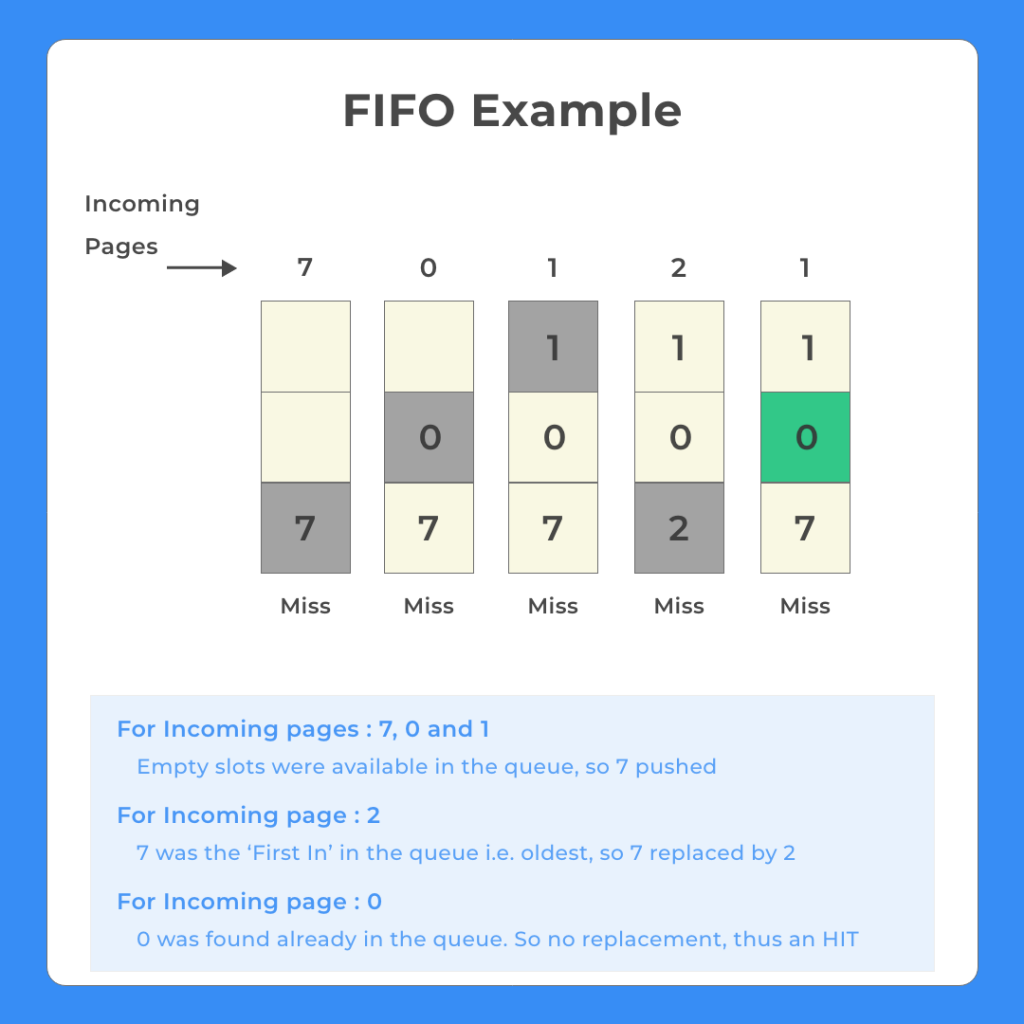
**FIFO-First In First Out**

A FIFO replacement algorithm associates with each page the time when that page was brought into memory.

This is how FIFO works –

If an incoming page is not available in any of the frames. Replacement shall be done.

Page replaced is according to FIFO (First in First Out)



The page that entered first must be swapped out first

In other words, the oldest frame must be swapped out

Page Replacement Algorithm In OS FIFO Example

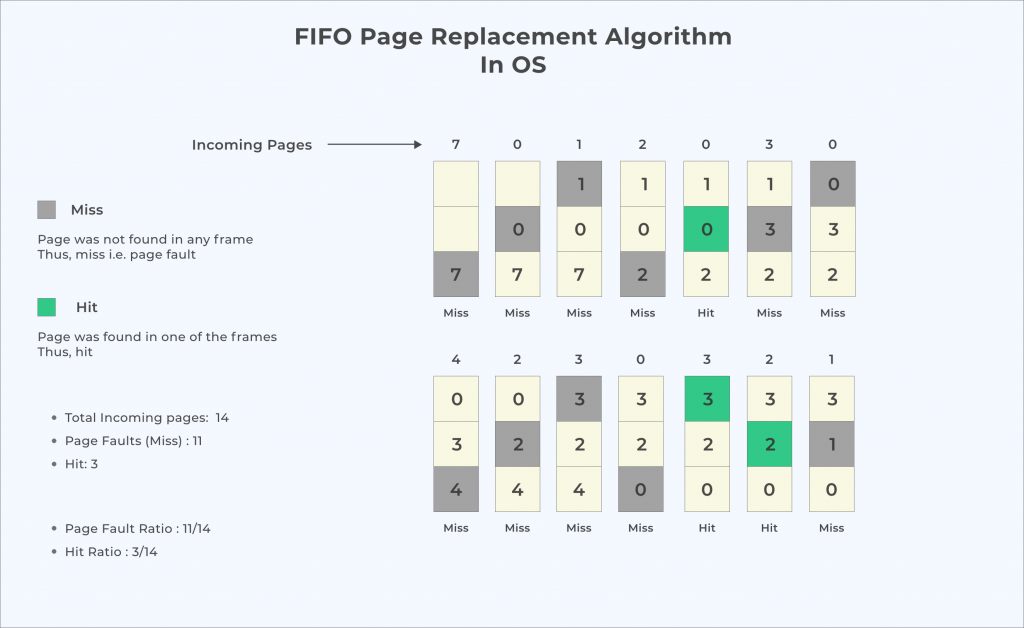
Example

It might be a confusing theory, lets look with an example –

Incoming page Steam – 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1

Page Size/ Frame Size = 3

Table will look like(Explanation after the table)



Explanation –

Step n [Incoming Stream] – Current Stack [x,y,z]

Current Oldest – In Red

After Replacing – In Green

Goes as

Step 1 [7] – Current Stack [nil, nil, nil] (Page Fault Occurs)

Result – [7, nil, nil]

Step 2 [0] – Current Stack [7, nil, nil]

Result – [7,0,nil] (Page Fault Occurs)

Step 3 [1] – Current Stack [7, 0, nil]

Result – [7,0,1] (Page Fault Occurs)

Step 4 [2] – Current Stack -[7,0,1]

Result – [2,0,1] (Page Fault Occurs)

Step 5 [0] – Current Stack [2,0,1]

Result – [2,0,1] as 0 is already Present

Step 6 [3] – Current Stack [2,0,1]

Result – [2,3,1]

Step 7 [0] – Current Stack [2,3,1]

Result – [2,3,0] (Page Fault Occurs)

Step 8 [4] – Current Stack [2,3,0]

Result – [4,3,0]

Step 9 [2] – Current Stack [4,3,0]

Result – [4,2,0] (Page Fault Occurs)

Step 10 [3] – Current Stack [4,2,0]

Result – [4,2,3] (Page Fault Occurs)

Step 11 [0] – Current Stack [4,2,3]

Result – [0,2,3] (Page Fault Occurs)

Step 12 [3] – Current Stack [0,2,3]

Result – [0,2,3] (3 already there)

Step 13 [Incoming Stream] – Current Stack [0,2,3]

Result – [0,2,3] (2 already there

Step 14 [1] – Current Stack [0,2,3]

Result – [0,1,3] (Page Fault Occurs)

If you have any questions ask in the comments section, and we will help.

Now page fault occurs 11 times out of 14.

So page fault ratio is 11/14.

**Belady’s Anomaly**

Think – Now, one should think that increase the page size(frame size as page size = frame size) will lead to less page fault, right!! since more chances of element to be present in the queue.

But, that’s not the case always. Sometimes by increasing the page size page fault rather increases, this type of anomaly is called belady’s Anomaly.(asked in AMCAT, CoCubes, Oracle)

For example, consider the following(solve yourself for practice)

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

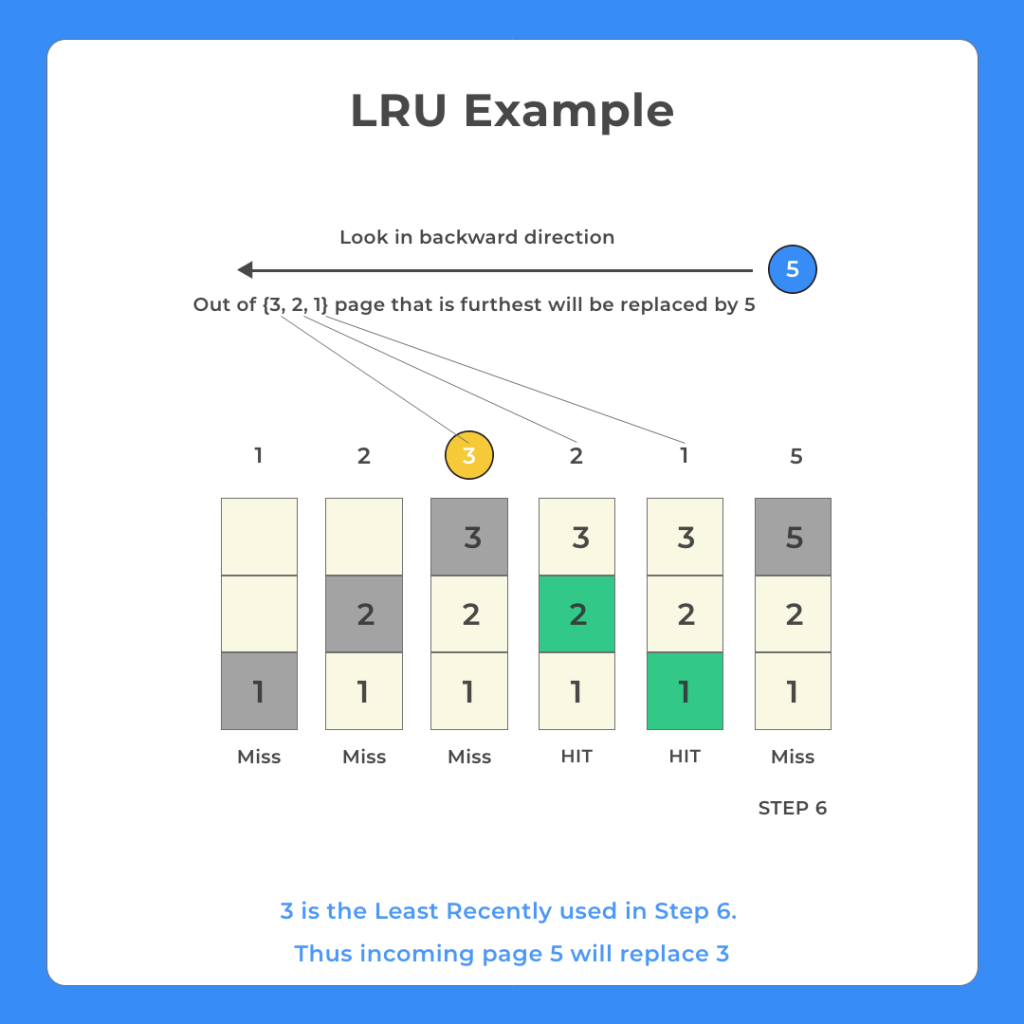
Using 3 slots gives 9-page faults

Using 4 slots gives 10-page faults

**Least Recently Used**

In this algorithm, we replace the element which is the current least recently used element in the current stack.

That is, when we look to the left of the table, that we have created we choose the further most page to get replaced.



Example

Let's say d is the incoming page and the current stack is [a,b,c].

Let's assume we are at the 10th iteration

1. a was last entered in current stack at 4th iteration.

2. b was entered at 2nd iteration

3. c was entered at 7th iteration

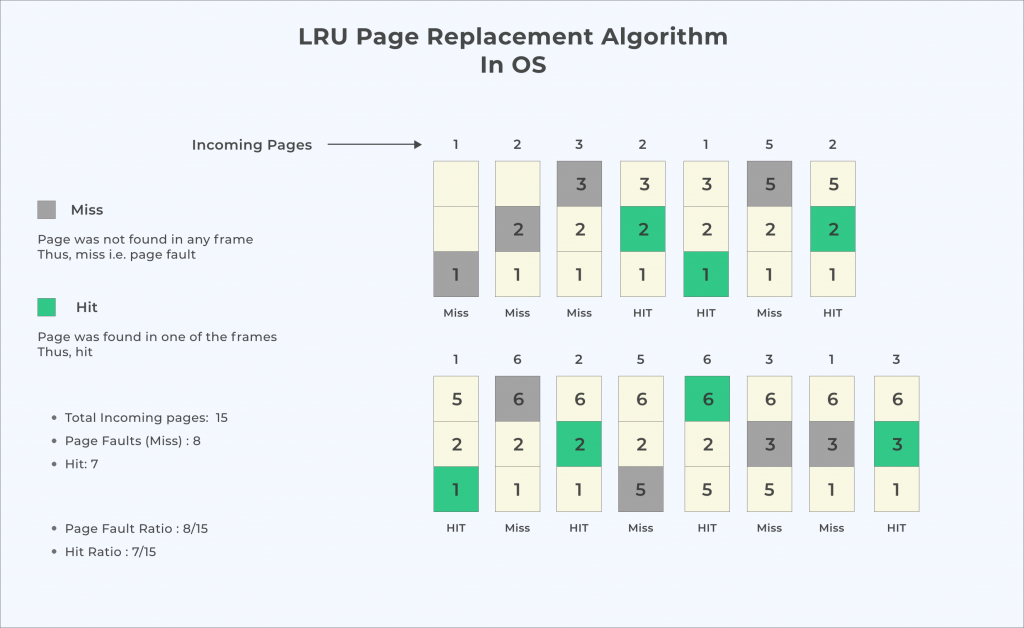
In this case d will replace b as b is the least recently used as was last seen in 2nd iteration

Page Replacement Algorithm In OS LRU Example

Trick

The trick is to look for the most recent occurrence of a page towards the left of the table and whichever is the furthermost. The incoming page should replace that.

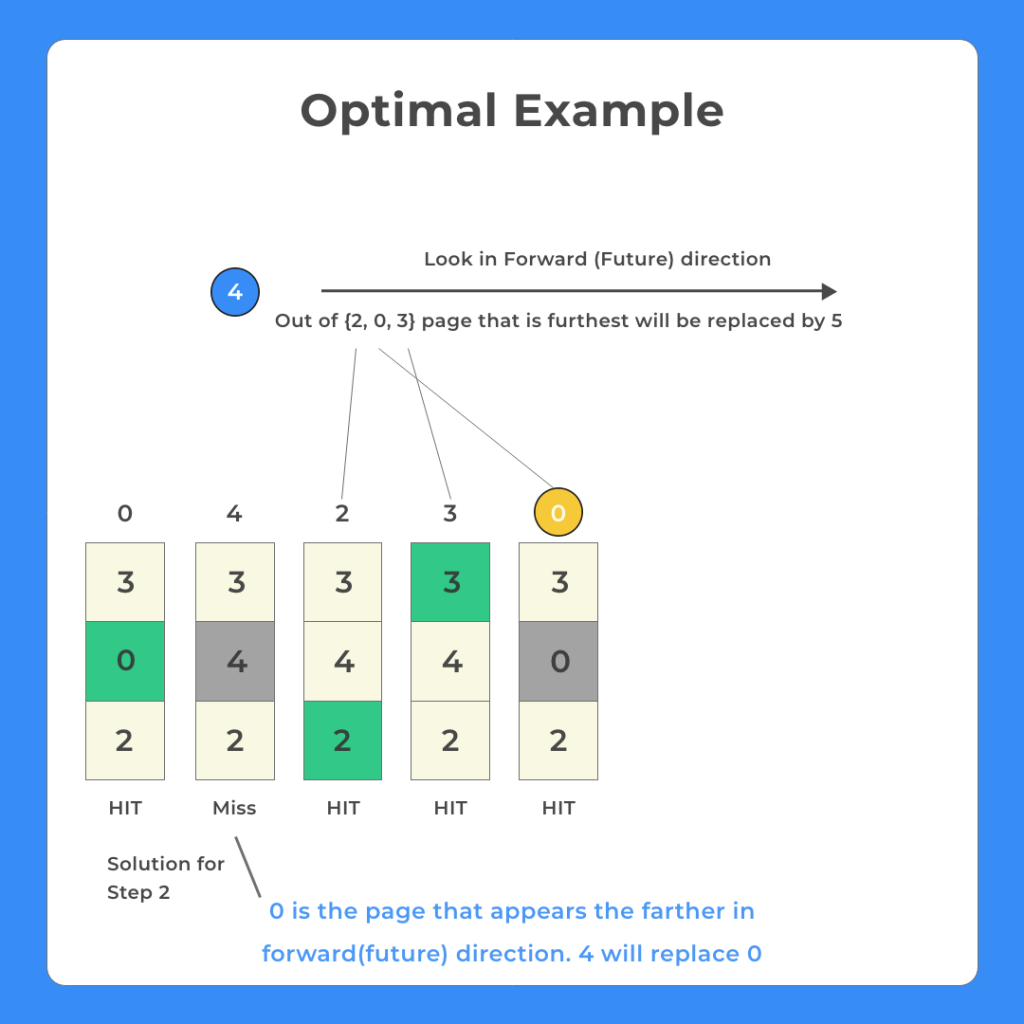
Page Replacement Algorithm LRU Example detailed



**Optimal Page Replacement**

In LRU we looked for the left furthermost page to replace. In optimal we do the opposite and look for right furthermost.

Now, this is done so that there are lesser page faults as the element will not used for the longest duration of time in the future.



Optimal Page Replacement Quick Facts

The result of the discovery of Belady’s Anamoly

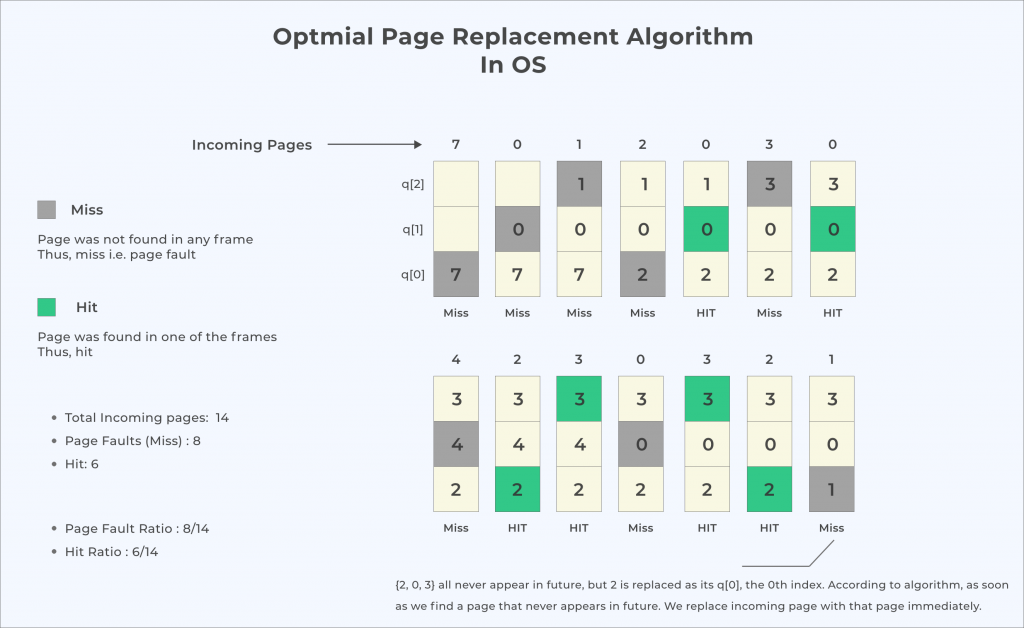
Lowest page fault rate of all algorithm’s and will never suffer from belady’s Anamoly.

Simply it replaces the pages that won’t be used for longest period of time.

Optimal page replacement is perfect, but not possible in practice as operating system cannot know future requests.

The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

The image below shows the implementation of the Optimal page replacement Algorithm.



**Least Frequently Used**

This algorithm is not for placement course.

But basically in current stack at any iteration we choose that element for replacement which has smallest count in the incoming page stream.

**Most Frequently Used**

This algorithm is not for placement course.

But basically in current stack at any iteration we choose that element for replacement which has highest count in the incoming page stream.

the page oldest page in the queue/memory will be replaced first.

Incoming page Steam – 7, 0, 1, 2, 0 , 3, 0, 4, 2, 3, 0, 3, 2, 1

Page Size/ Frame Size = 3

| **Incoming Page Stream ->** | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Existing at position 1 | 7 | 7 | 7 | 2 |  | 2 | 2 | 4 | 4 | 4 | 0 |  |  | 0 |
| Existing at position 2 |  | 0 | 0 | 0 |  | 3 | 3 | 3 | 2 | 2 | 2 |  |  | 1 |
| Existing at position 3 |  |  | 1 | 1 |  | 1 | 0 | 0 | 0 | 3 | 3 |  |  | 3 |
| Page Fault | x | x | x | x |  | x | x | x | x | x | x |  |  | x |

Method 1

Program in Java

// Code for FIFO page replacement algorithm in java

import java.util.HashSet;

import java.util.LinkedList;

import java.util.Queue;

class Main

{

static int pageFaults(int incomingStream[], int n, int frames)

{

System.out.println("Incoming \t Pages");

// Using Hashset to quickly check if a given

// incoming stream item in set or not

HashSet s = new HashSet<>(frames);

// Queue created to store pages in FIFO manner

// since set will not store order or entry

// we will use queue to note order of entry of incoming page

Queue queue = new LinkedList<>();

int page\_faults = 0;

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

{

// if set has lesser item than frames

if (s.size() < frames)

{

// If incoming item is not present, add to set

if (!s.contains(incomingStream[i]))

{

s.add(incomingStream[i]);

page\_faults++;

// Push the incoming page into the queue

queue.add(incomingStream[i]);

}

}

// If the set is full then we need to do page replacement

// in FIFO manner that is remove first item from both

// set and queue then insert incoming page

else

{

// If incoming item is not present

if (!s.contains(incomingStream[i]))

{

// remove the first page from the queue

int val = (int) queue.peek();

// remove from queue

queue.poll();

// Remove from set

s.remove(val);

// insert incoming page to set

s.add(incomingStream[i]);

// push incoming page to queue

queue.add(incomingStream[i]);

page\_faults++;

}

}

// printing happens here

System.out.print(incomingStream[i] + "\t");

System.out.print(queue + " \n");

}

return page\_faults;

}

public static void main(String args[])

{

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

int frames = 3;

int len = incomingStream.length;

int pageFaults = pageFaults(incomingStream, len, frames);

int hit = len - pageFaults;

System.out.println("Page faults: " + pageFaults);

System.out.println("Page fault Ratio: " + (double) pageFaults/len);

System.out.println("Hits: " + hit);

System.out.println("Hit Ratio : " + (double) hit/len);

}

Output

Incoming Pages

7 [7]

0 [7, 0]

1 [7, 0, 1]

2 [0, 1, 2]

3 [1, 2, 3]

0 [2, 3, 0]

4 [3, 0, 4]

2 [0, 4, 2]

3 [4, 2, 3]

0 [2, 3, 0]

1 [3, 0, 1]

Page faults: 11

Page fault Ratio: 0.7857142857142857

Hits: 3

Hit Ratio : 0.21428571428571427

Method 2

Program in Java

package LinkedLists;

// Code for FIFO page replacement algorithm in java

import java.util.HashSet;

import java.util.LinkedList;

import java.util.Queue;

class Main

{

public static void main(String args[])

{

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

int pageFaults = 0;

int frames = 3;

int m, n, s, pages;

pages = incomingStream.length;

System.out.println("Incoming \t Frame 1 \t Frame 2 \t Frame 3");

int temp[] = new int[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];

}

System.out.println();

System.out.print(incomingStream[m] + "\t\t\t");

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

{

if(temp[n] != -1)

System.out.print(temp[n] + "\t\t\t");

else

System.out.print(" - \t\t\t");

}

}

System.out.println("\nTotal Page Faults:\t" + pageFaults);

}

}

Output

Incoming Frame 1 Frame 2 Frame 3

7 7 - -

0 7 0 -

1 7 0 1

2 2 0 1

0 2 0 1

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 0 2 3

2 0 2 3

1 0 1 3

Total Page Faults: 11

**LRU in Java Language**

To implement this, a counter called an “age bit” is maintained, which keeps track of which page is to be referred and when it is to be referred. It ensures that the page which was least recently used is discarded to make space for the new page. When the page requested by the user is not present in the RAM, then a page fault occurs. When the page requested by the user is already present in the RAM, then page hit occurs. Let us understand LRU with an example.

Example

Reference String = 7 0 1 2 0 3 0 4 2 3

Number of iteration = 10

Page frame = 3

7 0 1 2 0 3 0 4 2 3

1 2 3 4 5 6 7 8 9 10

7 7 7 2 2 2 2 4 4 4

0 0 0 0 0 0 0 0 3

1 1 1 3 3 3 2 2

F F F F F F F F

Number of page fault = 8

Number of page hit = 2

**LRU Program in Java**

import java.util.\*;

class Lru

{

int p[],n,fr[],m,fs[],index,k,l,flag1=0,flag2=0,pf=0,framesize=3,i,j;

Scanner src=new Scanner(System.in);

void read()

{

System.out.println(“Enter page table size”);

n=src.nextInt(); p=new int[n];

System.out.println(“Enter the Reference String “);

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

p[i]=src.nextInt();

System.out.println(“Enter the Number of frames “);

m=src.nextInt();

fr=new int[m];

fs=new int[m];

}

void display()

{

System.out.println(“\n“);

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

{

if(fr[i]==-1)

System.out.println(“[ ]”);

else

System.out.println(“[“+fr[i]+“]”);

}

}

void lru()

{

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

{

fr[i]=-1;

}

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

{

flag1=0;flag2=0;

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

{

if(fr[i]==p[j])

{

flag1=1; flag2=1;

break;

}

}

if(flag1==0)

{

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

{

if(fr[i]==-1)

{

fr[i]=p[j];

flag2=1;

break;

}

}

}

if(flag2==0)

{

for(i=0;i<3;i++)

fs[i]=0;

for(k=j-1,l=1;l<=framesize-1;l++,k–)

{

for(i=0;i<3;i++)

{

if(fr[i]==p[k]) fs[i]=1;

}

}

for(i=0;i<3;i++)

{

if(fs[i]==0) index=i;

}

fr[index]=p[j];

pf++;

}

System.out.print(“Page : “+p[j]); display();

}

System.out.println(“\n Number of page fault:”+pf);

}

public static void main(String args[])

{

Lru a=new Lru();

a.read();

a.lru();

a.display();

}

}

Output

Enter page table size 10

Enter the Reference String:

1

3

1

2

6

2

8

1

3

1

Enter the Number of frames: 3

Page: 1

[1]

[]

[]

Page: 3

[1]

[3]

[]

Page: 1

[1]

[3]

[]

Page: 2

[1]

[3]

[2]

Page: 6

[1]

[6]

[2]

Page: 2

[1]

[6]

[2]

Page: 8

[8]

[6]

[2]

Page: 1

[8]

[1]

[2]

Page: 3

[8]

[1]

[3]

Page: 1

[8]

[1]

[3]

Number of page fault: 4

**Optimal Page Replacement**

The memory in an operating system is divided into pages. These pages are brought into the primary memory from the secondary memory as the CPU demands them. Pages that contain such critical data and information are swapped between different memory locations on the basis of their demand. To ensure that page which is not in frequent demand do not occupy critical memory space, the operating system sends it back to the secondary memory using a scheduling algorithm. The concept is also known as the Optimal Page Replacement technique.

**Optimal Page replacement program in JAVA**

// importing packages to use classes in the page replacement program

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class OptimalReplacement {

// creation of the main class to implement Optimal page replacement algorithm

public static void main(pagestring[] args) throws IOException

{

Countbuffer bfr = new Countbuffer(new InputStreamReader(System.in));

int frames, pointer = 0, hit = 0, fault = 0,strng\_size;

boolean isFull = false;

int buffer[];

int ref[];

int mem\_layout[][];

//Entering the number of frames

System.out.println(" Enter the total number of Frames: ");

frames = Integer.parseInt(br.readLine());

//Entering the string size of the reference

System.out.println(" Enter the reference string size:");

strng\_size = Integer.parseInt(br.readLine());

ref = new int[ref\_len];

mem\_layout = new int[strng\_size][frames];

buffer = new int[frames];

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

buffer[j] = -1;

//code to enter the reference string to carry out optimal page replacement

System.out.println(" Enter the reference string: ");

for(int i = 0; i < strng\_size; i++)

{

ref[i] = Integer.parseInt(br.readLine());

}

System.out.println();

for(int i = 0; i < strng\_size; i++)

{

int search = -1;

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

{

if(buffer[j] == ref[i])

{

search = j;

hit++;

break;

}

}

// code to update the stack checking its capacity

if(search == -1)

{

if(isFull)

{

int index[] = new int[frames];

boolean index\_flag[] = new boolean[frames];

for(int j = i + 1; j < ref\_len; j++)

{

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

{

if((ref[j] == buffer[k]) && (index\_flag[k] == false))

{

index[k] = j;

index\_flag[k] = true;

break;

}

}

}

//updating pointer to the correct memory location after checking capacity

buffer[pointer] = ref[i];

fault++;

if(!isFull)

{

pointer++;

if(pointer == frames)

{

pointer = 0;

isFull = true;

}

}

}

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

mem\_layout[i][j] = buffer[j];

}

// code to display the number strings

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

{

for(int j = 0; j < ref\_len; j++)

System.out.printf("%3d ",mem\_layout[j][i]);

System.out.println();

}

System.out.println("Hits: " + hit);

System.out.println("Hit Ratio: " + (float)((float)hit/str\_len));

System.out.println("Faults: " + fault);

}

}

Output

Enter the total number of Frames:

3

Enter the reference string size:

20

Enter the reference string:

1

2

3

2

1

5

2

1

6

2

5

6

3

1

3

6

1

2

4

3

1 1 1 1 1 1 1 1 6 6 6 6 6 6 6 6 6 2 4 4

-1 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1

-1 -1 3 3 3 5 5 5 5 5 5 5 3 3 3 3 3 3 3 3

Hits: 11

Hit Ratio: 0.55

Faults: 9