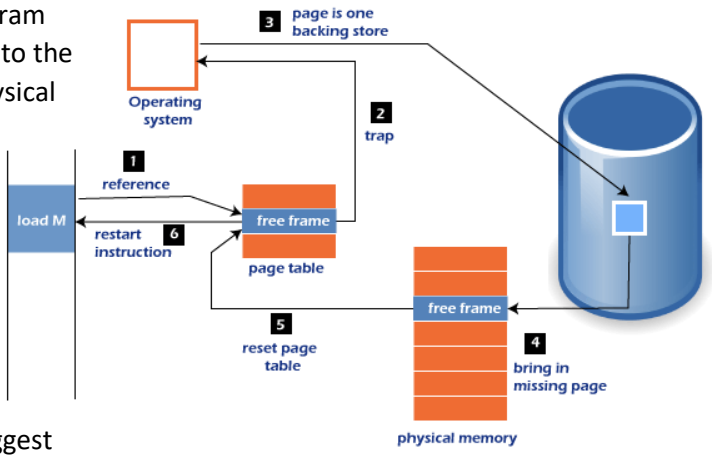
5rName:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	S. Y. B.Tech (Computer Engineering)
Course:	Operating System Laboratory
Course Code:	DJ19CEL403
Experiment No.:	09

AIM: **PAGE REPLACEMENT POLICIES (FIFO, LRU, OPTIMAL)**

THEORY:

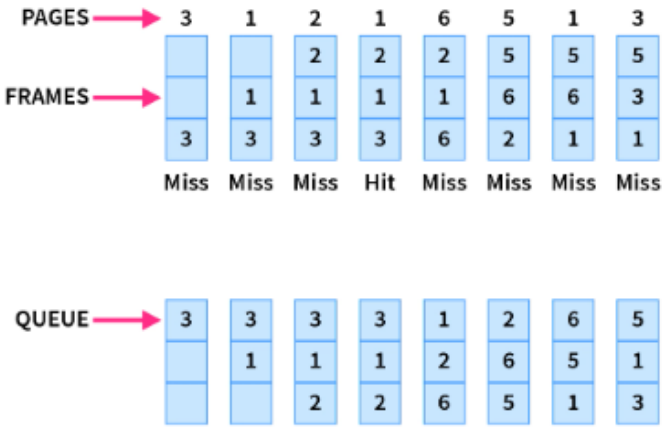
- In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.
- Page Fault
  - A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory.
  - Since actual physical memory is much smaller than virtual memory, page faults happen.
  - In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page.
  - Different page replacement algorithms suggest different ways to decide which page to replace.
  - The target for all algorithms is to reduce the number of page faults



FIFO

- FIFO algorithm is the simplest of all the page replacement algorithms.
- In this, we maintain a queue of all the pages that are in the memory currently.
- The oldest page in the memory is at the front-end of the queue and the most recent page is at the back or rear-end of the queue.
- Whenever a page fault occurs, the operating system looks at the front-end of the queue to know the page to be replaced by the newly requested page.
- It also adds this newly requested page at the rear-end and removes the oldest page from the front-end of the queue.

- Example: Consider the page reference string as 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames



CODE:

```
package Exp9;

public class FIFO{
    public static void main(String args[]){
        int referenceString[] = {3, 1, 2, 1, 6, 5, 1, 3};
        int pageFrame[] = {-1, -1, -1};

        int pageFaults = FIFO(referenceString, pageFrame);

        int pageHits = referenceString.length - pageFaults;

        System.out.println("Page Hits: "+pageHits);
        System.out.println("Page Faults: "+pageFaults);
        System.out.println("Page Hit Ratio:
"+(pageHits/referenceString.length));
        System.out.println("Page Fault Ratio:
"+(pageFaults/referenceString.length));
    }

    private static int FIFO(int referenceString[], int frames[]){
        int pageFaults = 0;
        int pointer = 0;
        for (int page = 0; page<referenceString.length; page++){

            boolean isPagePresent = false;
            for (int frame = 0; frame<frames.length; frame++){
                if (referenceString[page] == frames[frame]){
                    isPagePresent = true;
                }
```

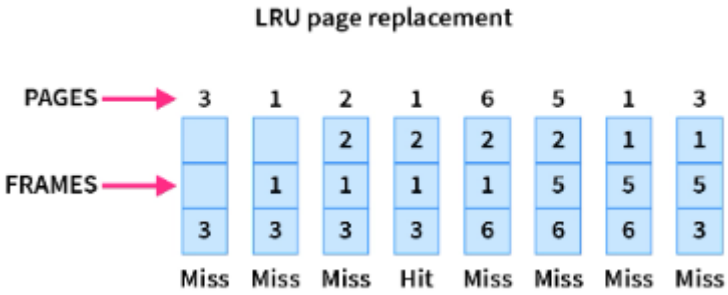
```
            }
            if (!isPagePresent){
                pageFaults++;
                frames[pointer] = referenceString[page];
                pointer++;
            }
            if (pointer==frames.length){
                pointer=0;
            }

            System.out.print(referenceString[page]+"\t\t\t | ");
            for (int i = 0;i<frames.length; i++){
                System.out.print(frames[i]+"\t | ");
            }
            System.out.println();
        }
        return pageFaults;
    }
}
```

OUTPUT:

```
29e69\redhat.java\jdt_ws\Code_68cc0323\bin' 'Exp9.FIFO'
3                       | 3     | -1    | -1    |
1                       | 3     | 1     | -1    |
2                       | 3     | 1     | 2     |
1                       | 3     | 1     | 2     |
6                       | 6     | 1     | 2     |
5                       | 6     | 5     | 2     |
1                       | 6     | 5     | 1     |
3                       | 3     | 5     | 1     |
Page Hits: 1
Page Faults: 7
Page Hit Ratio: 0
Page Fault Ratio: 0
PS C:\Users\Jadhav\Desktop\BTech\4th sem\OS\Prac\Code>
```

LRU

- The least recently used page replacement algorithm keeps the track of usage of pages over a period of time.
- This algorithm works on the basis of the principle of locality of a reference which states that a program tends to access the same set of memory locations repetitively over a short period of time.
- So pages that have been used heavily in the past are most likely to be used heavily in the future also.
- In this algorithm, when a page fault occurs, then the page that has not been used for the longest duration of time is replaced by the newly requested page.
- Example: Let's see the performance of the LRU on the same reference string of 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames:



CODE:

```
package Exp9;

import java.io.*;
import java.util.*;

public class LRU {

    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int frames,pointer = 0, hit = 0, fault = 0,ref_len;
        Boolean isFull = false;
        int buffer[];
        ArrayList<Integer> stack = new ArrayList<Integer>();
        int reference[] = {3, 1, 2, 1, 6, 5, 1, 3 };
        ref_len = reference.length;

        int mem_layout[][];

        frames = 3;

        mem_layout = new int[ref_len][frames];
        buffer = new int[frames];
        for(int j = 0; j < frames; j++)
            buffer[j] = -1;


        for(int i = 0; i < ref_len; i++)
        {
            if(stack.contains(reference[i]))
            {
             stack.remove(stack.indexOf(reference[i]));
```

```
            }
            stack.add(reference[i]);
            int search = -1;
            for(int j = 0; j < frames; j++)
            {
                if(buffer[j] == reference[i])
                {
                    search = j;
                    hit++;
                    break;
                }
            }
            if(search == -1)
            {
             if(isFull)
             {
              int min_loc = ref_len;
                    for(int j = 0; j < frames; j++)
                    {
                     if(stack.contains(buffer[j]))
                        {
                            int temp = stack.indexOf(buffer[j]);
                            if(temp < min_loc)
                            {
                                min_loc = temp;
                                pointer = j;
                            }
                        }
                    }
             }
                buffer[pointer] = reference[i];
                fault++;
                pointer++;
                if(pointer == frames)
                {
                 pointer = 0;
                 isFull = true;
                }
            }
            for(int j = 0; j < frames; j++)
                mem_layout[i][j] = buffer[j];
        }

        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("The number of Hits: " + hit);
        System.out.println("Hit Ratio: " + (float)((float)hit/ref_len));
        System.out.println("The number of Faults: " + fault);
    }

}
```

OUTPUT:

```
29e69\redhat.java\jdt_ws\Code_68cc0323\bin' 'Exp9.LRU'
  3   3   3   3   6   6   6   3
 -1   1   1   1   1   1   1   1
 -1  -1   2   2   2   5   5   5
The number of Hits: 2
Hit Ratio: 0.25
The number of Faults: 6
```

OPTIMAL

- Optimal page replacement is the best page replacement algorithm as this algorithm results in the least number of page faults.
- In this algorithm, the pages are replaced with the ones that will not be used for the longest duration of time in the future.
- In simple terms, the pages that will be referred farthest in the future are replaced in this algorithm.
- Example: Let's take the same page reference string 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames as we saw in FIFO.

CODE:

```
package Exp9;

class Optimal {
    static boolean search(int key, int[] fr)
    {
```

```
        for (int i = 0; i < fr.length; i++)
            if (fr[i] == key)
                return true;
        return false;
    }
    static int predict(int pg[], int[] fr, int pn,int index)
    {
        int res = -1, farthest = index;
        for (int i = 0; i < fr.length; i++) {
            int j;
            for (j = index; j < pn; j++) {
                if (fr[i] == pg[j]) {
                    if (j > farthest) {
                        farthest = j;
                        res = i;
                    }
                    break;
                }
            }

            if (j == pn)
                return i;
        }
        return (res == -1) ? 0 : res;
    }

    static void optimalPage(int pg[], int pn, int fn)
    {
        int[] fr = new int[fn];
        int hit = 0;
        int index = 0;
        for (int i = 0; i < pn; i++) {
            if (search(pg[i], fr)) {
                hit++;
                continue;
            }
            if (index < fn)
                fr[index++] = pg[i];
            else {
                int j = predict(pg, fr, pn, i + 1);
                fr[j] = pg[i];
            }
```

```
            System.out.print(pg[i]+"\t\t\t | ");
            for (int j = 0;j<fr.length; j++){
                System.out.print(fr[j]+"\t | ");
            }
            System.out.println();
        }
        System.out.println("No. of hits = " + hit);
        System.out.println("No. of misses = " + (pn - hit));
    }
    public static void main(String[] args)
    {

        int pg[]= { 3, 1, 2, 1, 6, 5, 1, 3  };
        int pn = pg.length;
        int fn = 4;
        optimalPage(pg, pn, fn);
    }
}
```

OUTPUT:

```
29e69\redhat.java\jdt_ws\Code_68cc0323\bin' 'Exp9.Optimal'
3                       | 3     | 0     | 0     | 0     |
1                       | 3     | 1     | 0     | 0     |
2                       | 3     | 1     | 2     | 0     |
6                       | 3     | 1     | 2     | 6     |
5                       | 3     | 1     | 5     | 6     |
No. of hits = 3
No. of misses = 5
PS C:\Users\Jadhav\Desktop\BTech\4th sem\OS\Prac\Code>
```

CONCLUSION:

- The objective of page replacement algorithms is to minimize the page faults
- FIFO page replacement algorithm replaces the oldest page in the memory
- Optimal page replacement algorithm replaces the page which will be referred farthest in the future
- LRU page replacement algorithm replaces the page that has not been used for the longest duration of time
- LIFO page replacement algorithm replaces the newest page in memory
- Random page replacement algorithm replaces any page at random
- Optimal page replacement algorithm is considered to be the most effective algorithm but cannot be implemented in practical scenarios due to various limitations