Name: Shardul Vikram Dharmadhikari
Roll no: 8016

ASSIGNMENT NO: 10

AIM:

Write a Java Program (using OOP features) to implement paging simulation using

1. Least Recently Used (LRU)

2. Optimal algorithm.

OBJECTIVES:

- To understand paging simulation and page replacement algorithms.
- To study Virtual Memory management techniques

PRE-REQUISITES:

1. Java Programming.
2. Basic of Operating

APPARATUS:

1. Java
2. Ubantu

THEORY:

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

The process address space is the set of logical addresses that a process references in its code. The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program.Virtual and physical addresses are the same in compile-

time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.
The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address..

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory. Paging is a **memory management technique** in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. Virtual memory is commonly implemented by **demand paging**.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program’s pages out to the disk or any of the new program’s pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

There are two types of page replacement algorithms

1. Least Recently Used(LRU)   2. Optimal Algorithm

1. Least Recently Used :-

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

2. Optimal algorithm:-

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Fig 2. Optimal Algorithm for Page replacement

CONCLUSION:

In page replacement Optimal Algorithm is more efficient than Least Recently Used algorithm because of less page faults.

CODE:

Lru.java:

```
/*
Problem Statement :
Write a Java Program (using OOP features) to implement paging simulation using
1. Least Recently Used (LRU)
2. Optimal algorithm
                                                  ****LRU****
*/
import java.io.*;
   class lru
    {
    public static void main(String args[])throws IOException
     {
                BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
                int f,page=0,ch,pgf=0,n,chn=0;
                boolean flag;
                int pages[];                //pgf-page fault

               System.out.println("1.LRU");
               int pt=0;
       System.out.println("enter no. of frames: ");
                f=Integer.parseInt(obj.readLine());
               int frame[]=new int[f];


               for(int i=0;i<f;i++)
               {
                       frame[i]=-1;
```

```
            }

            System.out.println("enter the no of pages ");
            n=Integer.parseInt(obj.readLine());

         pages=new int[n];
            System.out.println("enter the page no ");

            for(int j=0;j<n;j++)
            pages[j]=Integer.parseInt(obj.readLine());

            int pg=0;
            for(pg=0;pg<n;pg++)
    {
                    page=pages[pg];
                    flag=true;
                    for(int j=0;j<f;j++)
                    {
                            if(page==frame[j])
                            {
                                    flag=false;
                                    break;
                             }
                    }
                    int temp,h=3,i;
                    if(flag)
            {
                    if( frame[1]!=-1 && frame[2]!=-1 && frame[0]!=-1)
                            {
                                    temp=pages[pg-3];
                                    if(temp==pages[pg-2] || temp==pages[pg-1])
                                            temp=pages[pg-4];

                                    for(i=0;i<f;i++)
                                            if(temp==frame[i])
                                                    break;
                                    frame[i]=pages[pg];
                            }
                            else
                            {
                                    if(frame[0]==-1)
                                            frame[0]=pages[pg];
                                    else if(frame[1]==-1)
                                            frame[1]=pages[pg];
                                    else if(frame[2]==-1)
                                            frame[2]=pages[pg];
                            }


                            System.out.print("frame :");
                            for(int j=0;j<f;j++)
                            System.out.print(frame[j]+"   ");
                            System.out.println();
                            pgf++;
                    }
                    else
```

```
                              {
                                        System.out.print("frame :");
                                        for(int j=0;j<f;j++)
                                        System.out.print(frame[j]+"  ");
                                        System.out.println();
                              }

                    }//for

          System.out.println("Page fault:"+pgf);

}//main
}//class


/*
OUTPUT:-

1.LRU
enter no. of frames:
4
enter the no of pages
10
enter the page no
1
0
1
2
3
7
8
1
5
2
frame :1   -1   -1   -1
frame :1   0   -1   -1
frame :1  0  -1  -1
frame :1   0   2   -1
frame :1   3   2   -1
frame :7   3   2   -1
frame :7   3   8   -1
frame :7   1   8   -1
frame :5   1   8   -1
frame :5   1   2   -1
Page fault:9

*/
```

Optimal.java:

```
/*
Problem Statement :
Write a Java Program (using OOP features) to implement paging simulation using
1. Least Recently Used (LRU)
2. Optimal algorithm

                                                    ****Optimal****
*/
import java.util.*;
import java.io.*;


class Optimal
{
        public static void main(String args[])throws IOException
        {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                int numberOfFrames, numberOfPages, flag1, flag2, flag3, i, j, k, pos = 0, max;
                int faults = 0;
                int temp[] = new int[10];

                System.out.println("Enter number of Frames: ");
                numberOfFrames = Integer.parseInt(br.readLine());
                int frame[] = new int[numberOfFrames];


                System.out.println("Enter number of Pages: ");
                numberOfPages = Integer.parseInt(br.readLine());

                int pages[] = new int[numberOfPages];
                System.out.println("Enter the pages: ");
                for(i=0; i<numberOfPages; i++)
                        pages[i] = Integer.parseInt(br.readLine());

                for(i = 0; i < numberOfFrames; i++)
            frame[i] = -1;


                 for(i = 0; i < numberOfPages; ++i){
                     flag1 = flag2 = 0;

                     for(j = 0; j < numberOfFrames; ++j){
                        if(frame[j] == pages[i]){
                            flag1 = flag2 = 1;
                            break;
                         }
                     }

                     if(flag1 == 0){
                        for(j = 0; j < numberOfFrames; ++j){
                          if(frame[j] == -1){
```

```
                    faults++;
                    frame[j] = pages[i];
                    flag2 = 1;
                    break;
                }
            }
        }

        if(flag2 == 0){
            flag3 =0;

            for(j = 0; j < numberOfFrames; ++j){
                temp[j] = -1;

                for(k = i + 1; k < numberOfPages; ++k){
                    if(frame[j] == pages[k]){
                        temp[j] = k;
                        break;
                    }
                }
            }

            for(j = 0; j < numberOfFrames; ++j){
                if(temp[j] == -1){
                    pos = j;
                    flag3 = 1;
                    break;
                }
            }

            if(flag3 ==0){
                max = temp[0];
                pos = 0;

                for(j = 1; j < numberOfFrames; ++j){
                    if(temp[j] > max){
                        max = temp[j];
                        pos = j;
                    }
                }
            }

            frame[pos] = pages[i];
            faults++;
        }

//                      System.out.print();

        for(j = 0; j < numberOfFrames; ++j){
            System.out.print("\t"+ frame[j]);
        }
    }

    System.out.println("\n\nTotal Page Faults: "+ faults);
```

```
        }

}
```

O/p: //7 0 1 2 0 3 0 4 2 3 0 3 2