Operating System

Lab #4

(Page Replacement Policies)

★ Requirement:

Evaluating several page replacement algorithms.

- ★ FIFO “First in First out”
- ★ Optimal
- ★ LRU “least recently used’
- ★ Clock

★ Code:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define array_size 12


char type[20];

int *input; int

size;

int*buffer; int

n=0;
```

-This function is the implementation of FIFO replacement policy; it initializes the buffer and check if the new page referenced is already in the buffer or not. If not, it checks if the buffer has an empty place if it has then the page is simply put in the buffer, if not we increase the number of page faults and replaces the page with another one in the buffer

```
void FIFO()

{

   printf("Replacement Policy = %s\n",type);    printf("-----------------------------------

--\n");    printf("Page   Content of Frames\n");    printf("----   -----------------\n");

   int i,j=0,k,inBuffer,pagefault=0,emptyPlace=0,faultFlag=0;
```

```
  //initialize buffer
for(i=0; i<size; i++)
buffer[i]=-1;     for(i=0;
i<n; i++)
  {
      inBuffer=0;
      //If page exists in the buffer
for(k=0; k<size; k++)
      {
          if(buffer[k]==input[i])
inBuffer=1;
      }
      //if not in buffer
if(inBuffer==0)
      {
          //if buffer still has empty places
if(buffer[j]==-1)
emptyPlace=1;
buffer[j]=input[i];          j=(j+1)%size;
          //if no empty place then page fault
if(emptyPlace==0)
          {
              pagefault++;
faultFlag=1;

          }
      }
```

```
        if(faultFlag==1)
            printf("%02d F   ",input[i]);
    else
            printf("%02d     ",input[i]);
    for(k=0; k<size; k++)
        {
            if(buffer[k]!=-1)
    printf("%02d ",buffer[k]);
        }
        printf("\n");        //reset
    for next loop
    emptyPlace=faultFlag=0;
      }
      printf("-------------------------------------\n");
    printf("Number of page faults = %d\n",pagefault);
    }
```

-This function is the implementation of the OPTIMAL replacement policy; It looks forward in time to see which frame to replace on a page fault. Thus, it is not a real replacement algorithm. It gives us a frame of reference for a given static frame access sequence. Mainly, it checks the next value to see what is the most suitable page for replacement and what we will need later in order to keep it.

```
    void optimal()
    {
      printf("Replacement Policy = %s\n",type);    printf("-
    -------------------------------------\n");    printf("Page
    Content of Frames\n");    printf("----   -----------------
    \n");    int
i,j,k,p,inBuffer,pagefault=0,emptyPlace=0,temp[size],flag=0,max,faultFlag=0;
```

```
   //initialize buffer
for(i=0; i<size; i++)
buffer[i]=-1;

  for(i=0; i<n; i++)
    {
        inBuffer=0;
        //If page exists in the buffer
for(k=0; k<size; k++)
        {
            if(buffer[k]==input[i])
            {
                inBuffer=1;
            }
        }
        //if not in buffer
if(inBuffer==0)
        {

            for(k=0; k<size; k++)
            {   //if buffer still has empty places
if(buffer[k]==-1)
                {
                    buffer[k]=input[i];
emptyPlace=1;                    break;
                }
            }
            //if no empty place then page fault
if(emptyPlace==0)
           {   //check if a page is suitable for replacement
flag=0;
               //check if the numbers in the buffer will appear later or no
for(j=0; j<size; j++)
               {
                   temp[j]=-1;
                   //if it does save its position
for(k=i+1; k<n-1; k++)
                   {
                       if(buffer[j]==input[k])
                       {
temp[j]=k;
break;
                       }
                   }
               }
               for(j=0; j<size; j++)
               {
                   //if it doesn't appear later in the input
                   //save its position to replace it with another
page                       if(temp[j]==-1)
                   {
p=j;
flag=1;
break;
                   }
```

```
                }
                //if all of them will appear later
//choose one to remove from the buffer
                if(flag==0)
                {
                    max=temp[0];
p=0;
                    //save their positions in temp and remove it
for(j=1; j<size; j++)
                    {
                        if(temp[j]>max)
                        {
                            max=temp[j];
p=j;
                        }
                    }
                }
                buffer[p]=input[i];
pagefault++;                faultFlag=1;
            }
        }
        if(faultFlag==1)
printf("%02d F   ",input[i]);
else
            printf("%02d     ",input[i]);
for(k=0; k<size; k++)
        {
            if(buffer[k]!=-1)
      printf("%02d ",buffer[k]);
          }
          printf("\n");
          inBuffer=emptyPlace=flag=faultFlag=0;//reset for next loop
        }
        printf("-------------------------------------\n");
      printf("Number of page faults = %d\n",pagefault);
      }
```

-This function is the implementation of LRU replacement policy; On a page fault, the frame that was least recently used in replaced. A counter is kept to keep track of the number of times each page is used and the page with the minimum counter is then a candidate for replacement.

```
      void LRU()
      {
        printf("Replacement Policy = %s\n",type);
      printf("-------------------------------------\n");
      printf("Page   Content of Frames\n");
      printf("----   -----------------\n");     int
i,j,k,p,pagefault=0,inBuffer=0,temp[size],emptyPlace=0,min,counter=0,faultFlag=0;
        //initialize the buffer
      for(i=0; i<size; i++)
      buffer[i]=-1;     for(i=0;
      i<n; i++)
        {   //if page exist in buffer
          for(j=0; j<size; j++)
          {
            if(buffer[j]==input[i])
            {   //count every time the number appears
counter++;              temp[j]=counter;
inBuffer=1;             emptyPlace=1;
            }
        }
        //if not in buffer
if(inBuffer==0)
        {
            for(k=0; k<size; k++)
            {
                //if buffer still has empty places
if(buffer[k]==-1)
                {
                    counter++;
buffer[k]=input[i];
temp[k]=counter;
emptyPlace=1;                   break;
                }
            }
        }
         //else remove the LRU
if(emptyPlace==0)
        {
            min=temp[0];
p=0;
            //remove the page with the least counter,LRU page
for(j=1; j<size; j++)
            {
                if(temp[j]<min)
                {
                    min=temp[j];
p=j;
                }
            }
            counter++;
buffer[p]=input[i];
pagefault++;            faultFlag=1;
temp[p]=counter;
        }
        if(faultFlag==1)
printf("%02d F   ",input[i]);
else
            printf("%02d     ",input[i]);
for(k=0; k<size; k++)
        {
            if(buffer[k]!=-1)
printf("%02d ",buffer[k]);
        }
        inBuffer=emptyPlace=faultFlag=0;//reset for next loop printf("\n");
    }
    printf("-------------------------------------\n");
printf("Number of page faults = %d\n",pagefault);
}
```

-This function is the implementation of CLOCK replacement policy also known as “second Chance”. Mainly, When a page fault occurs and no empty frames exist, then the (secChance) (referenced) bit is inspected at the hand's location. If it is 0, the new page is put in place of the page the "hand" points to, and the hand is advanced one position. Otherwise, the (secChance) bit is cleared, then the clock hand is incremented and the process is repeated until a page is replaced

```
void clock()
{
   printf("Replacement Policy = %s\n",type);
printf("--------------------------------------\n");
printf("Page   Content of Frames\n");     printf("----
-----------------\n");
   int i,j=0,m=0,l,pagefault=0,inBuffer=0,secChance[size],fault=0;
   //initialize buffer and second chance
for(i=0; i<size; i++)
   {
      buffer[i]=-1;
secChance[i]=0;
   }
   for(i=0; i<n; i++)
   {
      //check if page exists in the buffer
for(j=0; j<size; j++)
      {
         if(buffer[j]==input[i])
         {
            inBuffer=1;
            secChance[j]=1;//set second chance bit
         }
      }
      //if page not in buffer
if(inBuffer==0)
```

```
        {
           while(inBuffer!=1)
           {
              //if buffer still has empty places
if(secChance[m]==0&&buffer[m]==-1)
              {
                 buffer[m]=input[i];
secChance[m]=1;
inBuffer=1;
              }
              else if(secChance[m]==0&&buffer[m]!=-1)
              {
                 buffer[m]=input[i];
secChance[m]=1;
inBuffer=1;
pagefault++;                  fault=1;
              }
              //buffer is full ,set the second chance to 0
              // to remove that page from buffer the next time
else
                 secChance[m]=0;
m=(m+1)%size;
           }




        }
        if(fault==1)
```

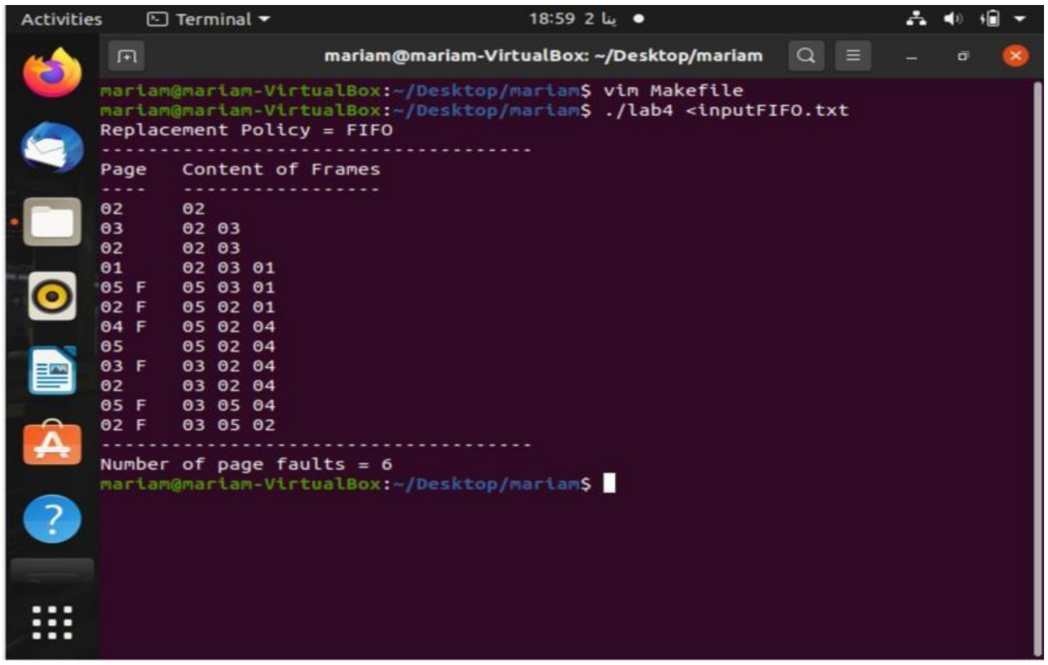
```
            printf("%02d F   ",input[i]);
        else
            printf("%02d     ",input[i]);
        for(l=0; l<size; l++)
          {
            if(buffer[l]!=-1)
        printf("%02d ",buffer[l]);
          }
          inBuffer=fault=0;//reset for next loop
        printf("\n");
         }
         printf("-------------------------------------\n");
        printf("Number of page faults = %d\n",pagefault);
        }
```

-The main function; it reads the file and takes the number of pages allocated to the process (size), and the type of replacement policy (type) and a simple while loop to call the function of the suitable replacement policy.

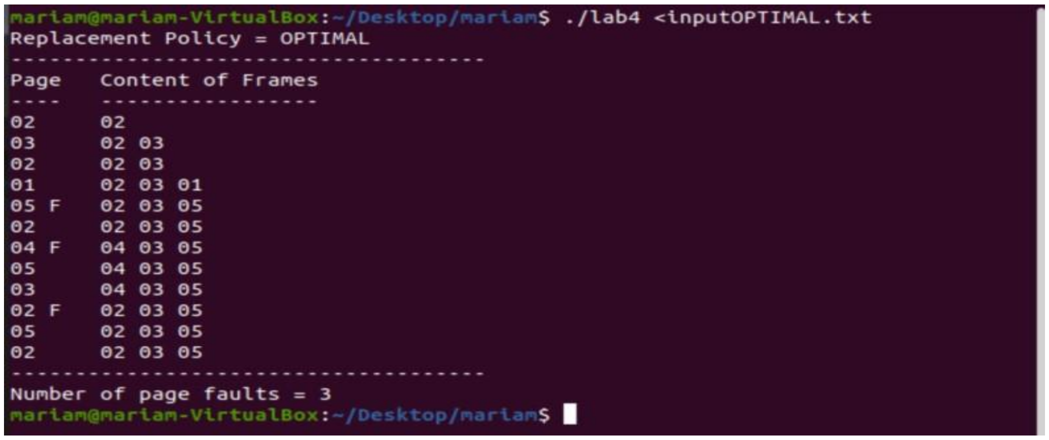
```
        int main()
        {
            scanf("%d",&size);
        scanf("%s",type);    int
        i=0;
          input=(int*)malloc(array_size*sizeof(int));
        scanf("%d",&input[i]);    while(input[i]!=-1)
          {
        i++;
            scanf("%d",&input[i]);
```

```
   }
n=i;
   buffer=(int*)malloc(size*sizeof(int));
if(strcmp(type,"CLOCK")==0)
       clock();
   else if(strcmp(type,"LRU")==0)
       LRU();
   else if(strcmp(type,"FIFO")==0)
       FIFO();
   else if(strcmp(type,"OPTIMAL")==0)
       optimal();
else
       printf("Please select a suitable page replacement!");
return 0;
}
```

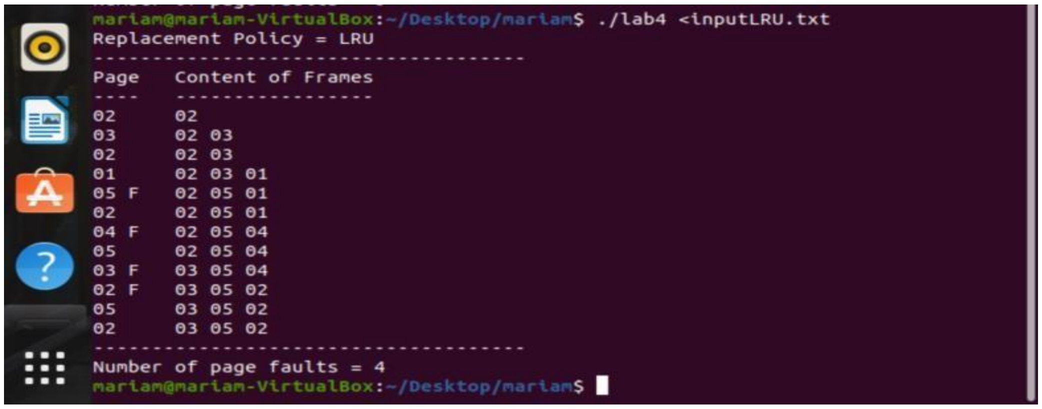
★ Results:



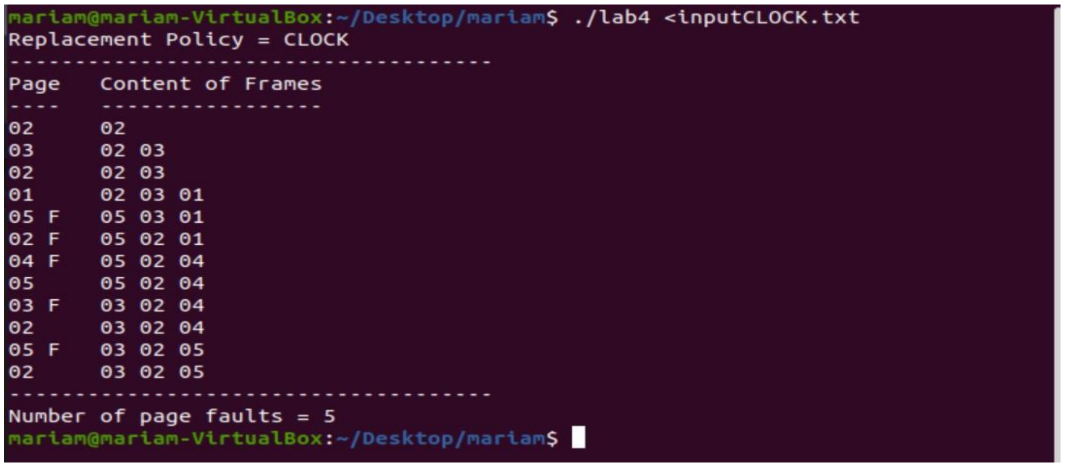
```
mariam@mariam-VirtualBox:~/Desktop/mariam$ vim Makefile
mariam@mariam-VirtualBox:~/Desktop/mariam$ ./lab4 <inputFIFO.txt
Replacement Policy = FIFO
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   05 03 01
02 F   05 02 01
04 F   05 02 04
05     05 02 04
03 F   03 02 04
02     03 02 04
05 F   03 05 04
02 F   03 05 02
-------------------------------------
Number of page faults = 6
mariam@mariam-VirtualBox:~/Desktop/mariam$ 
```



```
mariam@mariam-VirtualBox:~/Desktop/mariam$ ./lab4 <inputOPTIMAL.txt
Replacement Policy = OPTIMAL
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   02 03 05
02     02 03 05
04 F   04 03 05
05     04 03 05
03     04 03 05
02 F   02 03 05
05     02 03 05
02     02 03 05
-------------------------------------
Number of page faults = 3
mariam@mariam-VirtualBox:~/Desktop/mariam$ 
```



```
mariam@mariam-VirtualBox:~/Desktop/mariam$ ./lab4 <inputLRU.txt
Replacement Policy = LRU
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   02 05 01
02     02 05 01
04 F   02 05 04
05     02 05 04
03 F   03 05 04
02 F   03 05 02
05     03 05 02
02     03 05 02
-------------------------------------
Number of page faults = 4
mariam@mariam-VirtualBox:~/Desktop/mariam$ 
```



```
mariam@mariam-VirtualBox:~/Desktop/mariam$ ./lab4 <inputCLOCK.txt
Replacement Policy = CLOCK
-------------------------------------
Page   Content of Frames
----   -----------------
02     02
03     02 03
02     02 03
01     02 03 01
05 F   05 03 01
02 F   05 02 01
04 F   05 02 04
05     05 02 04
03 F   03 02 04
02     03 02 04
05 F   03 02 05
02     03 02 05
-------------------------------------
Number of page faults = 5
mariam@mariam-VirtualBox:~/Desktop/mariam$ 
```