EXPERIMENT - 5

Operating Systems Lab

AIM

Write a program for page replacement policy using:
a) LRU b) FIFO c) Optimal.

Syeda Reeha Quasar
14114802719
6C7

EXPERIMENT – 5

Aim:

Write a program for page replacement policy using:

a) LRU

b) FIFO

c) Optimal

Theory:

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault). There are various page replacement algorithms. Each algorithm has a different method by which the pages can be replaced.

a) **Least recent used (LRU) page replacement algorithm** → this algorithm replaces the page which has not been referred to for a long time. This algorithm is just the opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at the future.

b) **First In First Out (FIFO) page replacement algorithm** → in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on every page fault.

c) **Optimal Page Replacement algorithm** → this algorithm replaces the page which will not be referred to for so long in the future. Although it cannot be practically implementable it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.

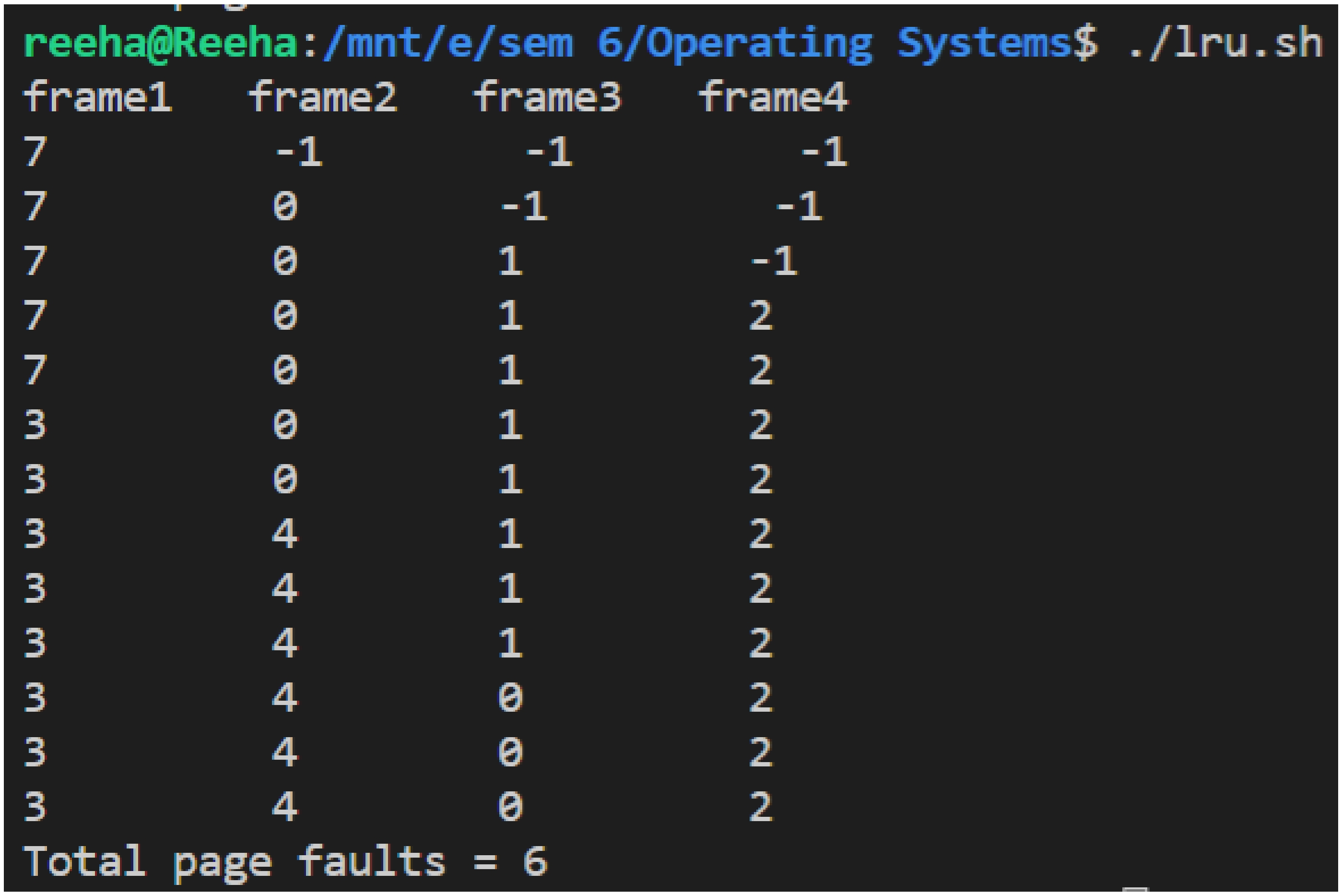
Source Code:

a) LRU (Least Recently Used) page replacement algorithm

```
#!/bin/bash
pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
capacity=4
count=0
fault=0
n=13
echo "frame1   frame2   frame3   frame4"
declare -a frames
declare -a time
i=0
while [ $i -lt $capacity ]
do
    frames[$i]=-1
    i=$((i+1))
```

```
done
j=0
while [ $j -lt $n ]
do
    flag1=0
    flag2=0
    k=0
    while [ $k -lt $capacity ]
    do
        if [ ${frames[$k]} -eq ${pages[$j]} ]
        then
            count=$((count+1))
            time[$j]=$count
            flag1=1
            flag2=1
            break
        fi
        k=$((k+1))
    done
    if [ $flag1 -eq 0 ]
    then
        a=0
        while [ $a -lt $capacity ]
        do
            if [ ${frames[$a]} -eq -1 ]
            then
                count=$((count+1))
                fault=$((fault+1))
                frames[$a]=${pages[$j]}
                time[$a]=$count
                flag2=1
                break
            fi
            a=$((a+1))
        done
    fi
    if [ $flag2 -eq 0 ]
    then
        pos=0
        minimum=${time[0]}
        b=1
        while [ $b -lt $capacity ]
        do
            if [ ${time[$b]} -lt $minimum ]
            then
                minimum=${time[$b]}
                pos=$b
            fi
            b=$((b+1))
        done
        count=$((count+1))
        fault=$((fault+1))
        frames[$pos]=${pages[$j]}
        time[$pos]=$count
    fi
    echo ${frames[0]} "       " ${frames[1]} "      " ${frames[2]} "       "
${frames[3]}
    j=$((j+1))
done
echo "Total page faults = 6"
```

Output:



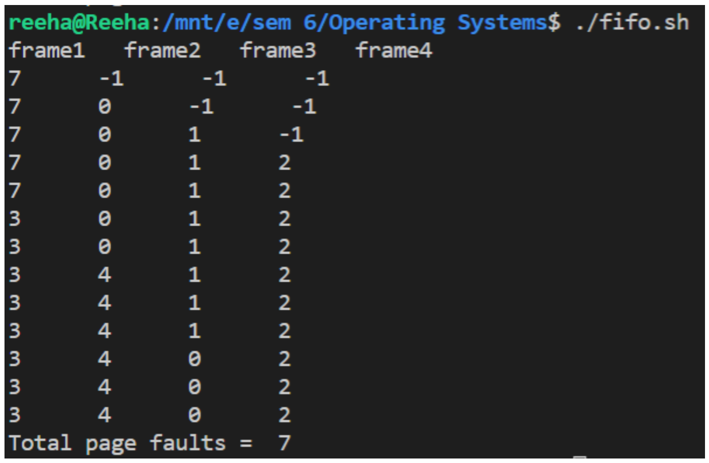
```
reeha@Reeha:/mnt/e/sem 6/Operating Systems$ ./lru.sh
frame1    frame2    frame3    frame4
7          -1         -1          -1
7          0         -1          -1
7          0         1          -1
7          0         1          2
7          0         1          2
3          0         1          2
3          0         1          2
3          4         1          2
3          4         1          2
3          4         1          2
3          4         0          2
3          4         0          2
3          4         0          2
Total page faults = 6
```

b) FIFO (First In First Out) page replacement algorithm

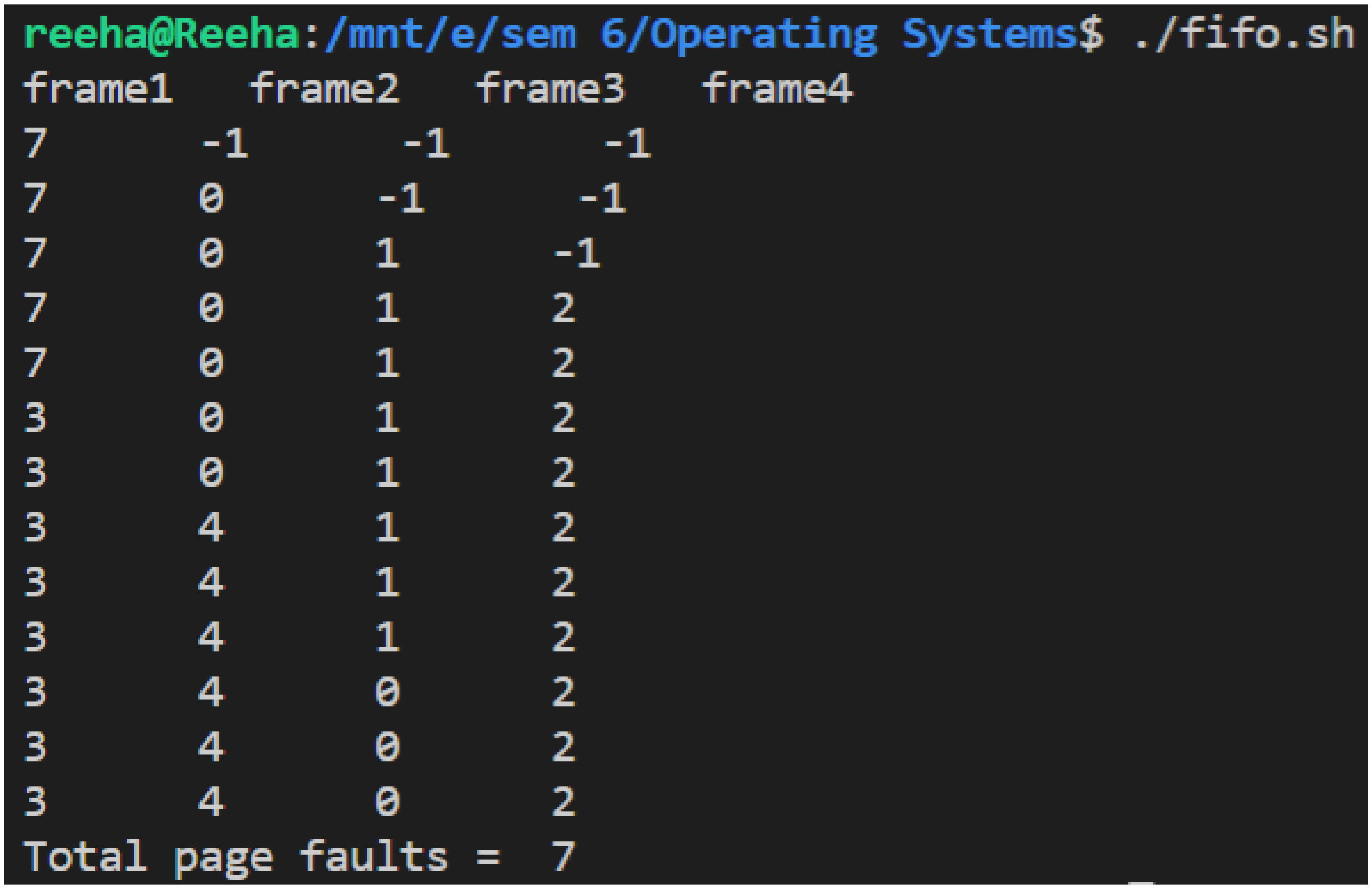
```
pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
capacity=4
count=0
fault=0
n=13
echo "frame1   frame2   frame3   frame4"
declare -a frames
i=0
```

```
while [ $i -lt $capacity ]
do
    frames[$i]=-1
    i=$((i+1))
done
j=0
while [ $j -lt $n ]
do
    flag=0
    k=0
    while [ $k -lt $capacity ]
    do
        if [ ${frames[$k]} -eq ${pages[$j]} ]
        then
            flag=1
            fault=$((fault-1))
        fi
        k=$((k+1))
    done
    fault=$((fault+1))
    if [ $fault -le $capacity ] && [ $flag -eq 0 ]
    then
        frames[$j]=${pages[$j]}
    else if [ $flag -eq 0 ]
    then
        frames[$(( (fault-1)%capacity))]=${pages[$j]}
    fi
    fi
    echo ${frames[0]} "    " ${frames[1]} "    " ${frames[2]} "    "
${frames[3]}
    j=$((j+1))
done
echo "Total page faults = " $fault
```

Output:



```
reeha@Reeha:/mnt/e/sem 6/Operating Systems$ ./fifo.sh
frame1   frame2   frame3   frame4
7      -1      -1      -1
7      0      -1      -1
7      0      1      -1
7      0      1      2
7      0      1      2
3      0      1      2
3      0      1      2
3      4      1      2
3      4      1      2
3      4      1      2
3      4      0      2
3      4      0      2
3      4      0      2
Total page faults =  7
```



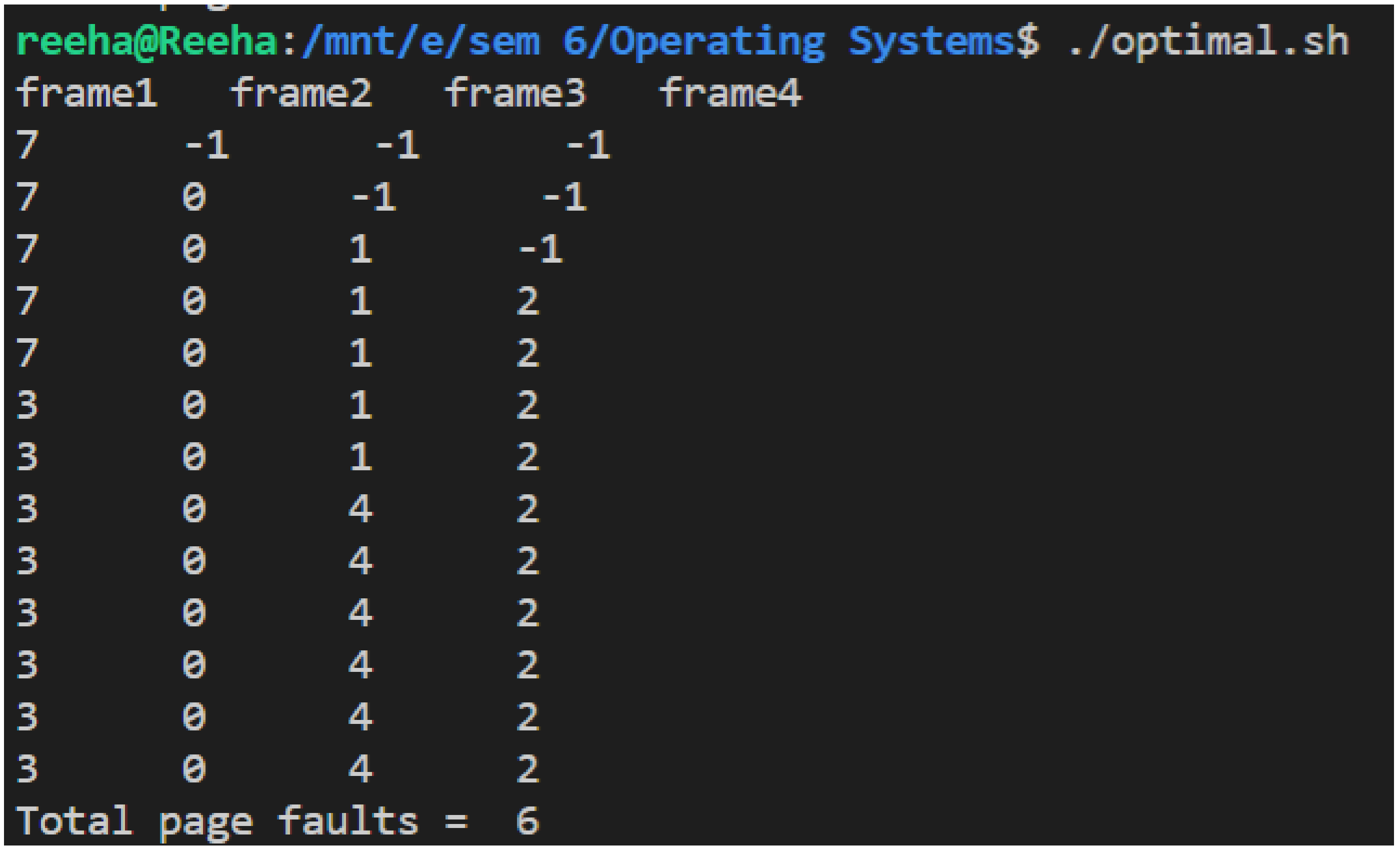
```
reeha@Reeha:/mnt/e/sem 6/Operating Systems$ ./fifo.sh
frame1    frame2    frame3    frame4
7       -1        -1        -1
7       0       -1        -1
7       0       1       -1
7       0       1       2
7       0       1       2
3       0       1       2
3       0       1       2
3       4       1       2
3       4       1       2
3       4       1       2
3       4       0       2
3       4       0       2
3       4       0       2
Total page faults =  7
```

c) Optimal page replacement algorithm

```
pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)
capacity=4
fault=0
n=13
declare -a frames
declare -a temp i=0
echo "frame1    frame2    frame3    frame4"
while [ $i -lt $capacity ]
do
    frames[$i]=-1
    i=$((i+1))
done
j=0
while [ $j -lt $n ]
do
    flag1=0
    flag2=0
    k=0
    while [ $k -lt $capacity ]
    do
        if [ ${frames[$k]} -eq ${pages[$j]} ]
        then
```

```
            flag1=1
            flag2=1
            break
        fi
        k=$((k+1))
    done
    if [ $flag1 -eq 0 ]
    then
        a=0
        while [ $a -lt $capacity ]
        do
            if [ ${frames[$a]} -eq -1 ]
            then
                fault=$((fault+1))
                frames[$a]=${pages[$j]}
                flag2=1
                break
            fi
            a=$((a+1))
        done
    fi
    if [ $flag2 -eq 0 ]
    then
        flag3=0
        x=0
        while [ $x -lt $capacity ]
        do
            temp[$x]=-1
            y=$((j+1))
            while [ $y -lt $n ]
            do
                if [ ${frames[$x]} -eq ${pages[$y]} ]
                then
                    temp[$x]=$y
                    break
                fi
                y=$((y+1))
            done
            x=$((x+1))
        done
        z=0
        while [ $z -lt $capacity ]
        do
            if [ ${temp[z]} -eq -1 ]
            then
                pos=$z flag3=1
                break
            fi
            z=$((z+1))
        done
        if [ $flag3 -eq 0 ]
        then
            maximum=${temp[0]}
            pos=0
            w=0
            while [ $w -lt $capacity ]
            do
                if [ ${temp[$w]} -gt $maximum ]
                then
                    maximum=${temp[$w]}
                    pos=$w
                fi
                w=$((w+1))
            done
        fi
        frames[$pos]=${pages[$j]}
        fault=$((fault+1))
    fi
    echo ${frames[0]} "    " ${frames[1]} "    " ${frames[2]} "    "
${frames[3]}
    j=$((j+1))
done
echo "Total page faults = " $fault
```

Output:



```
reeha@Reeha:/mnt/e/sem 6/Operating Systems$ ./optimal.sh
frame1   frame2   frame3   frame4
7      -1       -1       -1
7      0      -1      -1
7      0      1      -1
7      0      1      2
7      0      1      2
3      0      1      2
3      0      1      2
3      0      4      2
3      0      4      2
3      0      4      2
3      0      4      2
3      0      4      2
3      0      4      2
Total page faults =  6
```