**Experiment 11**

**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.

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

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

3. FIFO → 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.

**Source Code :**

1. **LRU:**

#!/bin/bash

pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)

capacity=4

count=0

fault=0

n=13

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

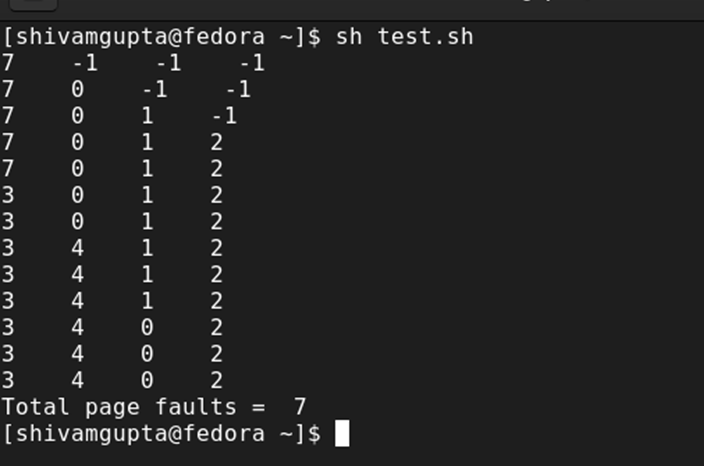
echo ${frames[0]} " " ${frames[1]} " " ${frames[2]} " " ${frames[3]}

j=$((j+1))

done

echo "Total page faults = " $fault

**Output :**

****

1. **FIFO:**

pages=(7 0 1 2 0 3 0 4 2 3 0 3 2)

capacity=4 count=0 fault=0 n=13

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

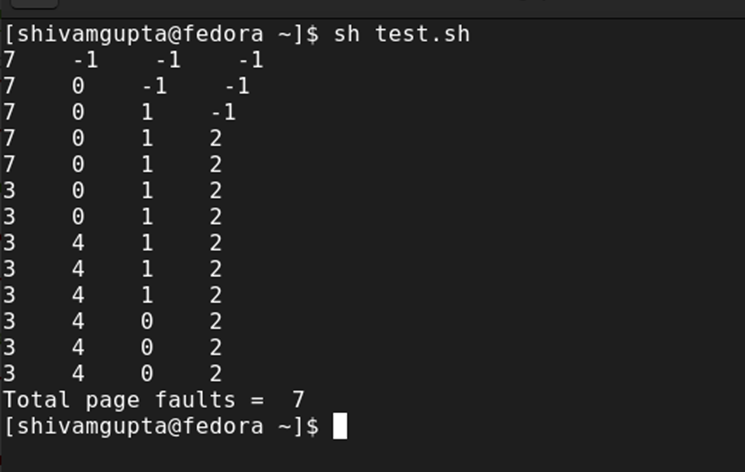
echo ${frames[0]} " " ${frames[1]} " " ${frames[2]} " " ${frames[3]}

j=$((j+1))

done

echo "Total page faults = " $fault

**Output :**

****

1. **Optimal:**

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

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

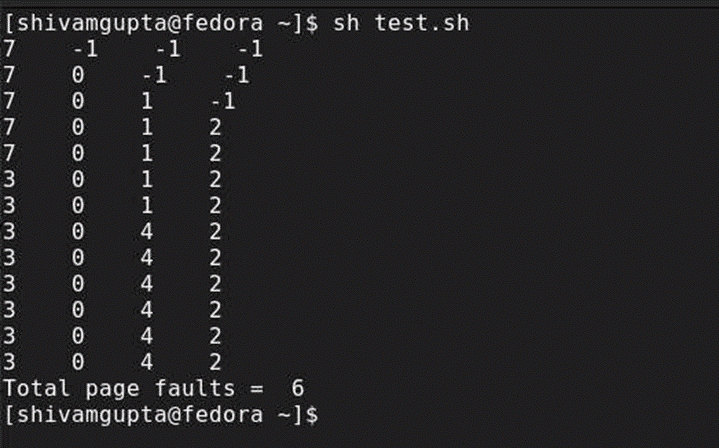
echo ${frames[0]} " " ${frames[1]} " " ${frames[2]} " " ${frames[3]}

j=$((j+1))

done

echo "Total page faults = " $fault

**Output :**

****

**Experiment 12**

**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.

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

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

3. FIFO → 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.

**Source Code :**

* **First Fit:**

#!/bin/bash

block=(100 500 200 300 600)

echo "Blocks = " ${block[@]}

process=(212 417 112 426)

m=5

n=4

declare -a allocation

i=0

while [ $i -lt $n ]

do

allocation[$i]=-1

i=$((i+1))

done

j=0

while [ $j -lt $n ]

do

k=0

while [ $k -lt $m ]

do

if [ ${block[$k]} -ge ${process[$j]} ]

then

allocation[$j]=$k

block[$k]=$((block[$k]-process[$j]))

break

fi

k=$((k+1))

done

j=$((j+1))

done

echo "Process No. Process Size Block No."

a=0

while [ $a -lt $n ]

do

if [ ${allocation[$a]} -ne -1 ]

then

echo " " $((a+1)) " " ${process[$a]} " " $((allocation[$a]+1))

else

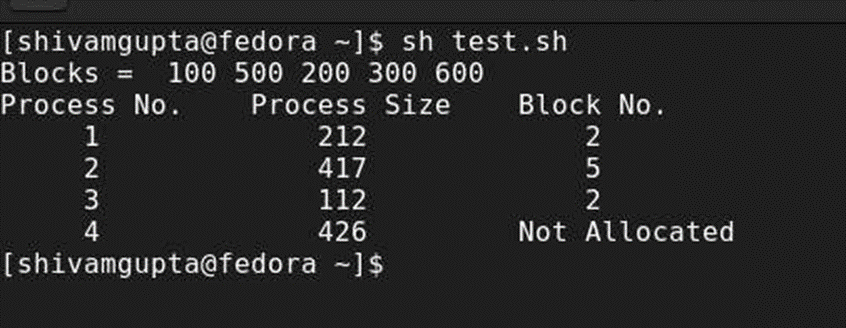
echo " " $((a+1)) " " ${process[$a]} " Not Allocated "

fi

a=$((a+1))

done

**Output :**

****

* **Best Fit:**

block=(100 500 200 300 600)

echo “Blocks = “ ${block[@]}

process=(212 417 112 426)

m=5

n=4

declare -a allocation

i=0

while [ $i -lt $n ]

do

allocation[$i]=-1

i=$((i+1))

done

j=0

while [ $j -lt $n ]

do

bestindex=-1

k=0

while [ $k -lt $m ]

do

if [ ${block[$k]} -ge ${process[$j]} ]

then

if [ $bestindex -eq -1 ]

then

bestindex=$k

else

if [ ${block[$bestindex]} -gt ${block[$k]} ]

then

bestindex=$k

fi

fi

fi

k=$((k+1))

done

if [ $bestindex -ne -1 ]

then

allocation[$j]=$bestindex

block[$bestindex]=$((block[$bestindex]-process[$j]))

fi

j=$((j+1))

done

echo "Process No. Process Size Block No."

a=0

while [ $a -lt $n ]

do

if [ ${allocation[$a]} -ne -1 ]

then

echo " " $((a+1)) " " ${process[$a]} " " $((allocation[$a]+1)) else

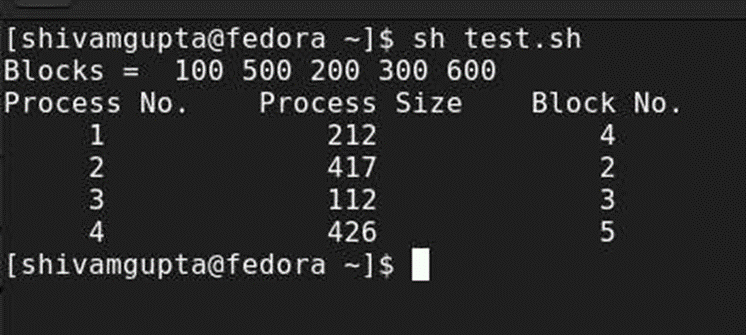
echo " " $((a+1)) " " ${process[$a]} " Not Allocated "

fi

a=$((a+1))

done

**Output :**

****

* **Worst Fit:**

block=(100 500 200 300 600)

echo “Blocks = “ ${block[@]}

process=(212 417 112 426)

m=5

n=4

declare -a allocation

i=0

while [ $i -lt $n ]

do

allocation[$i]=-1

i=$((i+1))

done j=0

while [ $j -lt $n ]

do

worstindex=-1

k=0

while [ $k -lt $m ]

do

if [ ${block[$k]} -ge ${process[$j]} ]

then

if [ $worstindex -eq -1 ]

then

worstindex=$k

else

if [ ${block[$worstindex]} -lt ${block[$k]} ]

then

worstindex=$k

fi

fi

fi

k=$((k+1))

done

if [ $worstindex -ne -1 ]

then

allocation[$j]=$worstindex

block[$worstindex]=$((block[$worstindex]-process[$j]))

fi

j=$((j+1))

done

echo “Process No. Process Size Block No.”

a=0

while [ $a -lt $n ]

do

if [ ${allocation[$a]} -ne -1 ]

then

echo “ “ $((a+1)) “ “ ${process[$a]} “ “ $((allocation[$a]+1))

else

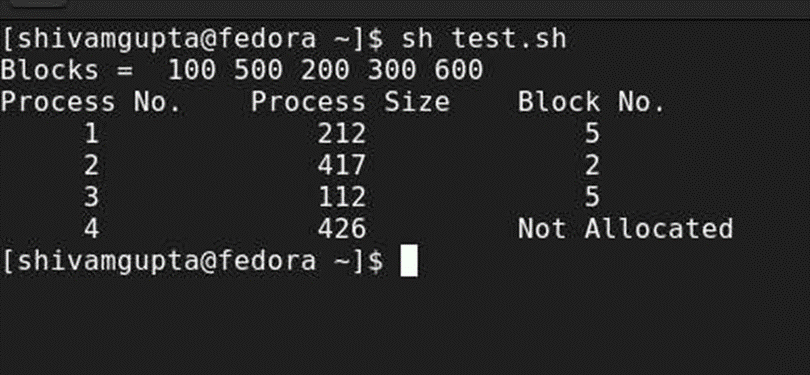
echo “ “ $((a+1)) “ “ ${process[$a]} “ Not Allocated”

fi

a=$((a+1))

done

**Output :**

****