Group Members:

Name : Sakshi Patil

Sap ID :60004180091

Name : Samiksha Patil

Sap ID :60004180092

**COA MINI PROJECT**

Topic:

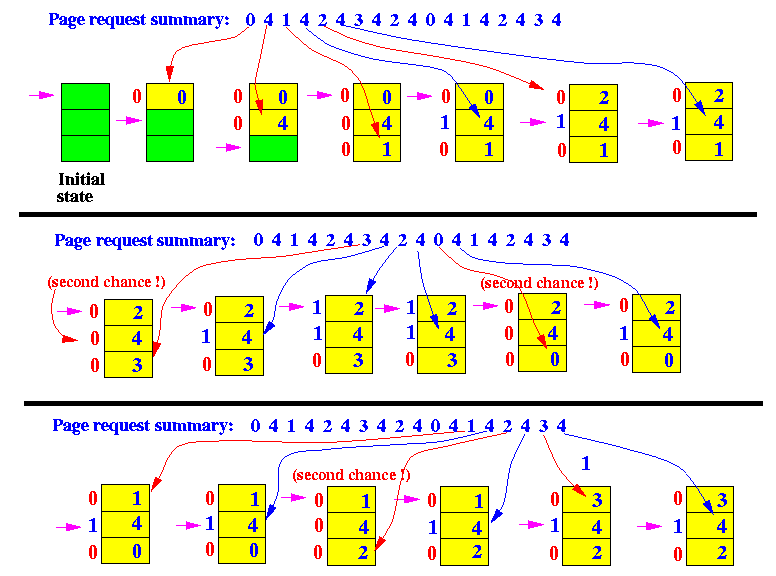
## The Second Chance Page Replacement Algorithm

A simple modification to FIFO that avoids the problem of throwing out a heavily used page is to inspect the *R*bit of the oldest page. If it is 0, the page is both old and unused, so it is replaced immediately. If the *R*bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory. Then the search continues.

**Implementation:**

* Add a "second chance" bit to each memory frame.
* Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance...
* A new page read into a memory frame has the second chance bit set to ZERO (0)
* When you need to find a page for removal, look in a round robin manner in the memory frames:
* If the second chance bit is ONE, reset its second chance bit (to ZERO) and continue.
* If the second chance bit is ZERO, replace the page in that memory frame.

The following figure shows the behavior of the program in paging using the Second Chance page replacement policy:



**Algorithm :**

Create an array frames to track the pages currently in memory and another Boolean array second\_chance to track whether that page has been accessed since it’s last replacement (that is if it deserves a second chance or not) and a variable pointer to track the target for replacement.

1. Start traversing the array arr. If the page already exists, simply set its corresponding element in second\_chance to true and return.
2. If the page doesn’t exist, check whether the space pointed to by pointer is empty (indicating cache isn’t full yet) – if so, we will put the element there and return, else we’ll traverse the array arr one by one (cyclically using the value of pointer), marking all corresponding second\_chance elements as false, till we find a one that’s already false. That is the most suitable page for replacement, so we do so and return.
3. Finally, we report the page fault count.

**Code :**

#include<stdio.h>

#define SIZE 3

int full=0;

int a[21];

int ref[SIZE];

int frame[SIZE];

int repptr=0;

int count=0;

int display()

{int i;

printf("\nThe elements in the frame are\n");

for(i=0;i<full;i++)

printf("%d\n",frame[i]);

}

int Pagerep(int ele)

{

int temp;

while(ref[repptr]!=0)

{ ref[repptr++]=0;

if(repptr==SIZE)

repptr=0;

}

temp=frame[repptr];

frame[repptr]=ele;

ref[repptr]=1;

return temp;

}

int Pagefault(int ele)

{if(full!=SIZE)

{ref[full]=1;

frame[full++]=ele;

}

else

printf("The page replaced is %d",Pagerep(ele));

}

int Search(int ele)

{int i,flag;

flag=0;

if(full!=0)

{

for(i=0;i<full;i++)

if(ele==frame[i])

{ flag=1;ref[i]=1;

break;

}}

return flag;

}

int main()

{

int n,i;

printf("The number of elements in the string are :");

scanf("%d",&n);

for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("\nThe elements present in the string are\n");

for(i=0;i<n;i++)

printf("%d ",a[i]);

printf("\n\n");

for(i=0;i<n;i++)

{

if(Search(a[i])!=1)

{

Pagefault(a[i]);

display();

count++;

}

}

printf("\nThe number of page faults are %d\n",count);

return 0;

}

**Output:**

