#### Page Replacement Algorithms

**Aim:**

Write a menu driven program for the implementation of the following Page

Replacement Algorithms:

(a) **FIFO** (b) **LRU** (c) **LFU**

**Description:**

**FIFO (First In First Out) Strategy**

The FIFO strategy is the simplest page replacement technique. It replaces the oldest page in memory when a new page needs to be loaded. This strategy uses a queue-like structure where pages are added to the end and removed from the front. In the code, pages are loaded sequentially into frames, and when a frame is full, the oldest page (the one that has been in memory the longest) is replaced by the new page. This method does not consider the frequency or recency of page usage, leading to potential inefficiencies known as Belady's anomaly, where increasing the number of frames can sometimes increase the number of page faults.

**Algorithm:**

1. start
2. Initialize j to 0, fcount (fault count) to 0, and avail (availability) to 0.
3. Print headers for the output.
4. For each page i from 0 to n-1:
   * Print the current page rs[i].
   * Set avail to 0.
   * For each frame k from 0 to nf-1:
     + If frames[k] equals rs[i], set avail to 1.
     + Print the current frame contents.
     + If avail is 1, print "H" (hit) and break.
   * If avail is 0:
     + Set frames[j] to rs[i].
     + Increment j (mod nf).
     + Increment fcount.
     + Print the current frame contents.
     + Print "F" (fault).
5. Print the total page fault count.
6. End

**LRU (Least Recently Used) Strategy**

The LRU strategy aims to replace the page that has not been used for the longest time, based on the idea that pages used recently are likely to be used again soon. The code implements this by maintaining a counter for each page to track how long it has been since the page was last used. When a page needs to be replaced, the page with the highest counter (i.e., the least recently used) is selected for replacement. This strategy generally performs better than FIFO because it considers the actual usage pattern of pages, though it requires additional overhead to maintain and update usage information for each page.

**Algorithm:**

1. start
2. Initialize fcount (fault count) to 0, k to 0, and c1 to 0.
3. Print headers for the output.
4. Set frames[k] to rs[0].
5. Print the initial frame state and increment fcount and k.
6. For each page i from 1 to n-1:
   * Print the current page rs[i].
   * Set c1 to 0.
   * For each frame j from 0 to nf-1:
     + If rs[i] is not in frames[j], increment c1.
   * If c1 equals nf (page fault):
     + Increment fcount.
     + If k is less than nf, set frames[k] to rs[i] and increment k.
     + Else:
       - Initialize c2 array to 0.
       - For each frame r from 0 to nf-1:
         * For each previous page j from i-1 to 0:

If frames[r] is not equal to rs[j], increment c2[r].

Else, break.

* + - * Copy c2 to b and sort b in descending order.
      * Set the frame with the highest count (b[0]) to rs[i].
    - Print the current frame contents and "F" (fault).
  + Else, print the current frame contents and "H" (hit).

1. Print the total page fault count.
2. end

**LFU (Least Frequently Used) Strategy**

The LFU strategy replaces the page that has been used the least number of times. It keeps track of the frequency of access for each page, assuming that pages with lower access frequencies are less likely to be used again in the near future. The code maintains a frequency count for each page and sorts pages based on their frequencies. When a page fault occurs, the page with the lowest frequency is replaced. This strategy can effectively handle scenarios where certain pages are frequently accessed, but it might struggle with pages that have fluctuating access patterns, as it does not consider the recency of accesses.

**Algorithm:**

1. start
2. Initialize front to 0, fcount (fault count) to 0.
3. Print headers for the output.
4. For each page i from 0 to n-1:
   * Print the current page rs[i].
   * If rs[i] is a page fault and front is greater than or equal to nf:
     + Find the index with the minimum frequency using find\_min\_freq().
     + Set frames[index] to rs[i].
     + Increment fcount.
     + Print the current frame contents and "F" (fault).
   * Else if rs[i] is a page fault:
     + Set frames[front] to rs[i].
     + Increment front and fcount.
     + Print the current frame contents and "F" (fault).
   * Else if rs[i] is not a page fault:
     + Print the current frame contents and "H" (hit).
5. Print the total page fault count.
6. end

**Program:**

#include<stdio.h>

typedef struct {

int page;

int freq;

} Freq;

void FIFO(int rs[],int n,int frames[],int nf);

void LRU(int rs[],int n,int frames[],int nf);

void LFU(int rs[],int n,int frames[],int nf);

int check\_fault(int page,int frames[],int nf);

int find\_min\_freq(int frames[],int current,int nf,int rs[]);

void sort(Freq table[],int len);

void main(){

int n,frames[50],rs[50],nf,i;

printf("Number of pages: ");

scanf("%d",&n);

printf("Enter the page numbers : \n");

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

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

}

printf("Number of frames: ");

scanf("%d",&nf);

printf("\n---------------------------------------------------------");

printf("\nMenu\n1.FIFO\n2.LRU\n3.LFU\n4.Exit");

int choice;

do{

for(i=0;i<nf;i++){

frames[i]=-1;

}

printf("\nEnter choice: ");

scanf("%d",&choice);

switch(choice){

case 1: FIFO(rs,n,frames,nf); break;

case 2: LRU(rs,n,frames,nf); break;

case 3: LFU(rs,n,frames,nf); break;

case 4: break;

}

}while(choice!=4);

}

void FIFO(int rs[],int n,int frames[],int nf){

int j=0,k,fcount=0,avail;

printf("\nCurrentPage\tPages in Frames");

for(int i=1;i<nf;i++) printf("\t");

printf("Hit/Fault\n");

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

printf("%d\t\t",rs[i]);

avail=0;

for(k=0;k<nf;k++){

if(frames[k]==rs[i]){

avail=1;

for(k=0;k<nf;k++){

if(frames[k]==-1) printf("X\t");

else printf("%d\t",frames[k]);

}

printf("H");

}

}

if(avail==0){

frames[j]=rs[i];

j=(j+1)%nf;

fcount++;

for(k=0;k<nf;k++){

if(frames[k]==-1) printf("X\t");

else printf("%d\t",frames[k]);

}

printf("F");

}

printf("\n");

}

printf("Page fault count = %d",fcount);

}

void LRU(int rs[],int n,int frames[],int nf){

int fcount=0,k=0;

int c1,i,j,c2[100],b[100],t,r;

printf("\nCurrentPage\tPages in Frames");

for(int i=1;i<nf;i++) printf("\t");

printf("Hit/Fault\n");

frames[k]=rs[k];

printf("%d\t\t",rs[0]);

for(i=0;i<nf;i++){

if(frames[i]==-1) printf("X\t");

else printf("%d\t",frames[i]);

}

printf("F\n");

fcount++;

k++;

for(i=1;i<n;i++){

printf("%d\t\t",rs[i]);

c1=0;

for(j=0;j<nf;j++){

if(rs[i]!=frames[j]) c1++;

}

if(c1==nf){

fcount++;

if(k<nf){

frames[k]=rs[i];

k++;

for(j=0;j<nf;j++){

if(frames[j]==-1) printf("X\t");

else printf("%d\t",frames[j]);

}

printf("F\n");

}

else{

for(r=0;r<nf;r++){

c2[r]=0;

for(j=i-1;j>=0;j--){

if(frames[r]!=rs[j]) c2[r]++;

else break;

}

}

for(r=0;r<nf;r++){

b[r]=c2[r];

}

for(r=0;r<nf;r++){

for(j=r;j<nf;j++){

if(b[r]<b[j]){

t=b[r];

b[r]=b[j];

b[j]=t;

}

}

}

for(r=0;r<nf;r++){

if(c2[r]==b[0]) frames[r]=rs[i];

if(frames[r]==-1) printf("X\t");

else printf("%d\t",frames[r]);

}

printf("F\n");

}

}

else{

for(j=0;j<nf;j++){

if(frames[j]==-1) printf("X\t");

else printf("%d\t",frames[j]);

}

printf("H\n");

}

}

printf("the no. of page fault is %d\n",fcount);

}

void sort(Freq table[],int len){

for(int i=0;i<len;i++){

for(int j=i;j<len-i-1;j++){

if (table[j].freq>table[j+1].freq){

Freq temp;

temp=table[j];

table[j]=table[j+1];

table[j+1]=temp;

}

}

}

}

int check\_fault(int page,int frames[],int nf){

for(int i=0;i<nf;i++){

if (frames[i]==page){

return 0;

}

}

return 1;

}

int find\_min\_freq(int frames[],int current,int nf,int rs[]){

Freq table[30];

int count;

for(int i=0;i<nf;i++){

table[i].page=frames[i];

}

for(int i=0;i<nf;i++){

count=0;

for(int j=0;j<current;j++){

if (rs[j]==table[i].page){

count++;

}

}

table[i].freq=count;

}

sort(table,nf);

for(int i=0;i<nf;i++){

if(frames[i]==table[0].page){

return i;

}

}

return -1;

}

void LFU(int rs[],int n,int frames[],int nf){

int front=0;

int fcount=0;

int j;

printf("\nCurrentPage\tPages in Frames");

for(int i=1;i<nf;i++) printf("\t");

printf("Hit/Fault\n");

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

printf("%d\t\t",rs[i]);

if(check\_fault(rs[i],frames,nf) && front>=nf){

int index=find\_min\_freq(frames,i,nf,rs);

frames[index]=rs[i];

fcount++;

for(j=0;j<nf;j++){

if(frames[j]==-1) printf("X\t");

else printf("%d\t",frames[j]);

}

printf("F\n");

}

else if(check\_fault(rs[i],frames,nf)){

frames[front]=rs[i];

front++;

fcount++;

for(j=0;j<nf;j++){

if(frames[j]==-1) printf("X\t");

else printf("%d\t",frames[j]);

}

printf("F\n");

}

else if(!check\_fault(rs[i],frames,nf)){

for(j=0;j<nf;j++){

if(frames[j]==-1) printf("X\t");

else printf("%d\t",frames[j]);

}

printf("H\n");

}

}

printf("the no. of page fault is %d\n",fcount);

}

**Output:**

**PS D:\Coding\repositories\S4-OSlab> gcc .\PageReplacement.c**

**PS D:\Coding\repositories\S4-OSlab> ./a.exe**

**Number of pages: 8**

**Enter the page numbers :**

**1 2 4 3 4 1 2 1**

**Number of frames: 3**

**---------------------------------------------------------**

**Menu**

**1.FIFO**

**2.LRU**

**3.LFU**

**4.Exit**

**Enter choice: 1**

**CurrentPage Pages in Frames Hit/Fault**

**1 1 X X F**

**2 1 2 X F**

**4 1 2 4 F**

**3 3 2 4 F**

**4 3 2 4 H**

**1 3 1 4 F**

**2 3 1 2 F**

**1 3 1 2 H**

**Page fault count = 6**

**Enter choice: 2**

**CurrentPage Pages in Frames Hit/Fault**

**1 1 X X F**

**2 1 2 X F**

**4 1 2 4 F**

**3 3 2 4 F**

**4 3 2 4 H**

**1 3 1 4 F**

**2 2 1 4 F**

**1 2 1 4 H**

**the no. of page fault is 6**

**Enter choice: 3**

**CurrentPage Pages in Frames Hit/Fault**

**1 1 X X F**

**2 1 2 X F**

**4 1 2 4 F**

**3 3 2 4 F**

**4 3 2 4 H**

**1 1 2 4 F**

**2 1 2 4 H**

**1 1 2 4 H**

**the no. of page fault is 5**

**Enter choice: 4**

**Exitted**

**Result:**

The program has been executed and output has been verified.