**LAB 8**

**11.Write a C program to simulate page replacement algorithms**

**a) FIFO**

**b) LRU**

**c) Optimal**

#include<stdio.h>

int n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data)

{

hit=0;

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

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void fifo()

{

initialize();

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

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

for(k=0; k<nf-1; k++)

p[k]=p[k+1];

p[k]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void optimal()

{

initialize();

int near[50];

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

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

near[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

near[j]=9999;

}

int max=-9999;

int repindex;

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

{

if(near[j]>max)

{

max=near[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void lru()

{

initialize();

int least[50];

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

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

for(k=i-1; k>=0; k--)

{

if(pg==in[k])

{

least[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

least[j]=-9999;

}

int min=9999;

int repindex;

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

{

if(least[j]<min)

{

min=least[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault!");

}

dispPgFaultCnt();

}

int main()

{

int choice;

while(1)

{

printf("\nPage Replacement Algorithms\n1.Enter data\t2.FIFO\t3.Optimal\t4.LRU\t5.Exit\nEnter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

getData();

break;

case 2:

fifo();

break;

case 3:

optimal();

break;

case 4:

lru();

break;

default:

return 0;

break;

}

}

}





***12.Write a C program to simulate the following file allocation strategies.***

***a) Sequential***

***b) Indexed***

***c) Linked***

**SEQUENTIAL**

#include<stdio.h>

struct fileTable

{ char name[20];

int sb, nob;}ft[30];

void main()

{

int i, j, n;

char s[20];

printf("Enter no of files :");

scanf("%d",&n);

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

{

printf("\nEnter file name %d :",i+1);

scanf("%s",ft[i].name);

printf("Enter starting block of file %d :",i+1);

scanf("%d",&ft[i].sb);

printf("Enter no of blocks in file %d :",i+1);

scanf("%d",&ft[i].nob);

}

printf("\nEnter the file name to be searched -- ");

scanf("%s",s);

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

if(strcmp(s, ft[i].name)==0)

break;

if(i==n)

printf("\nFile Not Found");

else

{

printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");

printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);

for(j=0;j<ft[i].nob;j++)

printf("%d, ",ft[i].sb+j);

}

}



**LINKED**

#include<stdio.h>

struct fileTable

{

char name[20]; int nob; struct block \*sb;

}ft[30];

struct block

{

int bno;

struct block \*next;

};

void main()

{

int i, j, n;

char s[20];

struct block \*temp;

printf("Enter no of files :");

scanf("%d",&n);

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

{

printf("\nEnter file name %d :",i+1);

scanf("%s",ft[i].name);

printf("Enter no of blocks in file %d :",i+1);

scanf("%d",&ft[i].nob);

ft[i].sb=(struct block\*)malloc(sizeof(struct block));

temp = ft[i].sb;

printf("Enter the blocks of the file :");

scanf("%d",&temp->bno);

temp->next=NULL;

for(j=1;j<ft[i].nob;j++)

{

temp->next = (struct block\*)malloc(sizeof(struct block)); temp = temp->next;

scanf("%d",&temp->bno);

}

temp->next = NULL;

}

printf("\nEnter the file name to be searched -- ");

scanf("%s",s);

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

if(strcmp(s, ft[i].name)==0)

break;

if(i==n)

printf("\nFile Not Found");

else

{

printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");

printf("\n%s\t\t%d\t",ft[i].name,ft[i].nob);

temp=ft[i].sb;

for(j=0;j<ft[i].nob;j++)

{

printf("%d -> ",temp->bno);

temp = temp->next;

}}}



**INDEXED**

#include <stdio.h>

#include <conio.h>

#include <string.h>

struct fileTable

{

char name[20];

int nob, blocks[30];

} ft[30];

void main()

{

int i, j, n;

char s[20];

printf("Enter no of files :");

scanf("%d", &n);

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

{

printf("\nEnter file name %d :", i + 1);

scanf("%s", ft[i].name);

printf("Enter no of blocks in file %d :", i + 1);

scanf("%d", &ft[i].nob);

printf("Enter the blocks of the file :");

for (j = 0; j < ft[i].nob; j++)

scanf("%d", &ft[i].blocks[j]);

}

printf("\nEnter the file name to be searched -- ");

scanf("%s", s);

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

if (strcmp(s, ft[i].name) == 0)

break;

if (i == n)

printf("\nFile Not Found");

else

{

printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");

printf("\n %s\t\t%d\t", ft[i].name, ft[i].nob);

for (j = 0; j < ft[i].nob; j++)

printf("%d, ", ft[i].blocks[j]);

}

}

