Page Replacement Algorithms:

Code:

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

//Global Variables below:

int optArr[100];

int pageRefStr[100];//={1,2,1,3,7,4,5,6,3,1};

//{4,7,6,1,7,6,1,2,7,2};

//{1,2,3,4,2,1,5,3,2,4,6};

//

int pageFaults=0;

//

typedef struct node

{

int val;

int sizeAlloc; //no. of frames allocated in the memory for this process.

struct node \*next;

}node;

node \*rt=NULL;

void getPageRefStr()

{

int sizeRefStr =0;

printf("\nEnter the size of the page reference string: ");

scanf("%d",&sizeRefStr);

printf("\n\nEnter the page reference string: ");

int i=0;

while(sizeRefStr>0)

{

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

sizeRefStr--;

i++;

}

pageRefStr[i]='\0';

printf("\n");

}

node \*init(node \*rt,int n) //initialize the linked list

{

//set pagefault = 0

pageFaults=0;

//init. llist below

while(n>0)

{

node \*new\_node=(node \*)malloc(sizeof(node));

new\_node->val=0;

new\_node->next=NULL;

if(rt==NULL)

{

rt=new\_node;

rt->sizeAlloc=n;

}

else

{

node \*p=rt;

while(p->next!=NULL)

{

p=p->next;

}

p->next=new\_node;

}

n--;

}

return rt;

}//einit

void display(node \*rt)

{

node \*ptr=rt;

while(ptr->next!=NULL)

{

printf("%d ",ptr->val);

ptr=ptr->next;

}

printf("%d\n",ptr->val);

}//edisplay.

int isValPresentAlready(node \*rt,int search)

{

node \*p=rt;

while(p!=NULL)

{

if(p->val==search)

{

return 1;

}

p=p->next;

}

return 0; //otherwise

}//epresent

node \*fifo(node \*rt)

{

int n=0;

n=rt->sizeAlloc;

int i=0;

int counter=0;

int present=0;

node \*p=rt;

printf("\n");

for(i=0;pageRefStr[i]!='\0';i++)

{

if(i==0)

{

printf("Stage 0 \n");

}

present=isValPresentAlready(rt,pageRefStr[i]);

if(present)

{

continue;

}

if(p==NULL)

{

//replace a page in the memory (linked list)

p=rt;

p->val=pageRefStr[i];

printf("\nStage %d\n",++counter);

printf("%d ",p->val);

p=p->next;

pageFaults++;

continue;

}

if(!present)

{

p->val=pageRefStr[i];

printf("%d ",p->val);

pageFaults++;

p=p->next;

}

}

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

return rt;

}//efifo

node \*findLru(int \*lru,int currIndex,int framesAllocated,node \*rt)

{

\*lru=currIndex-framesAllocated;

node \*p=rt ;

while(p!=NULL)

{

if(p->val==pageRefStr[\*lru])

{

return p;

}

p=p->next;

}

}//efindLru

node \*lru(node \*rt)

{

int n=0;

n=rt->sizeAlloc;

int flag=0;

int i=0;

int present=0,counter=0;

node \*p=rt;

for (i = 0; pageRefStr[i]!='\0'; i += 1)

{

if(i==0)

{

printf("Stage 0 \n");

}

present=isValPresentAlready(rt,pageRefStr[i]);

if(present)

{

continue;

}

if(p==NULL ||flag==1)

{

//replace a page in the memory (linked list)

int lru=0;

//printf("\nEntered Null");

node \*indexOfLru=(node \*)malloc(sizeof(node));

indexOfLru=findLru(&lru,i,n,rt);

p=indexOfLru;

p->val=pageRefStr[i];

printf("\nStage %d\n",++counter);

display(rt);

p=p->next;

pageFaults++;

flag=1;

continue;

}

if(!present)

{

p->val=pageRefStr[i];

printf("%d ",p->val);

pageFaults++;

p=p->next;

}

}

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

return rt;

}//elru

/\*int uniqueSet(int arr[],int n=0)

{

int dum[100]=0;

for()

}//eunique\*/

int optSearch(node \*rt,int dummy[],int currIndex)

{

int n=rt->sizeAlloc;

int count=0;

n=rt->sizeAlloc;

//making Prediction

for(int j=0;j<n;j++)

{

for(int i=currIndex+1;pageRefStr[i]!='\0' && count!=n-1;i++)

{

if(dummy[j]==pageRefStr[i])

{

//printf("\ndummy[%d] = %d",j,dummy[j]);

dummy[j]=-1;

count++;

break;

}

}

}

//return the element that will not be used in the near future

for(int x=0;x<rt->sizeAlloc;x++)

{

if(dummy[x]!=-1)

{

return dummy[x];

}

}

return dummy[0];

}//eoptsearch

node \*findOptIndex(node \*rt,int search)

{

node \*p=rt;

while(p!=NULL)

{

if(p->val==search)

return p;

p=p->next;

}

}//efindOptIndex

node \*optimal(node \*rt)

{

//Assuming the pageRefStr will not be unique, else in unique values case use fifo/any other algorithm.

node \*p=rt;

int i=0,counter=0;

int present=0;

int elm=0; //this is the element to be replaced

printf("\n");

int flag=0;

int dummy[rt->sizeAlloc];

int index=0;

node \*ptr=p;

for(i=0;pageRefStr[i]!='\0';i++)

{

while(ptr!=NULL)

{

dummy[index]=ptr->val;

//printf("\ndummy[%d] = %d\n\n",index,dummy[index]);

index++;

ptr=ptr->next;

}

if(i==0)

{

printf("Stage 0 \n");

}

present=isValPresentAlready(rt,pageRefStr[i]);

if(present)

{

continue;

}

if(p==NULL || flag==1)

{

//replace a page in the memory (linked list)

flag=1;

node \*indexOfElm;

elm=optSearch(rt,dummy,i);

indexOfElm=findOptIndex(rt,elm);

p=indexOfElm;

p->val=pageRefStr[i];

printf("\nStage %d\n",++counter);

//displaying ram below

display(rt);

//end of display

p=p->next;

pageFaults++;

index=0;

ptr=rt; //setting the ptr to always point to the top of the ram after every iteration;

continue;

}

if(!present)

{

//printf("\nval = %d\n",pageRefStr[i]);

p->val=pageRefStr[i];

printf("%d ",p->val);

pageFaults++;

p=p->next;

}

index=0;

ptr=rt; //setting the ptr to always point to the top of the ram after every iteration;

}

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

return rt;

}

int main()

{

int n=0;

getPageRefStr();

printf("\nEnter the no. of frames to be \nallocated in the RAM for this process : ");

scanf("%d",&n);

//n=3;

rt=init(rt,n);

int ch=0;

while(1==1)

{

printf("\n\nPress:\n\n1:FIFO\n2:LRU (LEAST RECENTLY USED)\n3:Optimal Algorithm\n4:Any other no. to exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:rt=fifo(rt);

rt=NULL;

rt=init(rt,n);

break;

case 2:rt=lru(rt);

rt=NULL;

rt=init(rt,n);

break;

case 3:rt=optimal(rt);

rt=NULL;

rt=init(rt,n);

break;

default:

printf("\nTerminating the program\n");

exit(0);

}//eswitch

}//ewhile outter

}//emg

Output:



