**WEEK 9**

**Yapara karthikeya**

**1BM21CS249**

**16-08-2023**

**Q. Write a C program to simulate disk scheduling algorithms**

1. **FCFS**
2. **SCAN**
3. **C-SCAN**
4. **SSTF**
5. **LOOK**
6. **C-LOOK**

#include <stdio.h>

#include <stdlib.h>

int queue[10];

int n;

int head;

void sort(int arr[]){

int t;

for(int i=0;i<n-1;i++){

for(int j=0;j<n-i-1;j++){

if(arr[j+1]<arr[j]){

t=arr[j+1];

arr[j+1]=arr[j];

arr[j]=t;

}

}

}

return ;

}

int fcfs(){

int move=0;

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

{

move+=abs(head-queue[i]);

head=queue[i];

}

return move;

}

int sstf(){

int move=0;

int vis[n];

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

{

vis[i]=0;

}

int count=0;

int u;

while(count!=n){

int min=999;

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

if(abs(head-queue[i])<min && vis[i]==0){

u=i;

min = abs(head-queue[i]);

}

}

vis[u]=1;

move+=abs(head-queue[u]);

head=queue[u];

count++;

}

return move;

}

int scan(){

int move=0;

int l,u,d;

printf("enter the lower and upper limit:\n");

scanf("%d%d",&l,&u);

printf("enter the direction:(1->UP and 0->down):\n");

scanf("%d",&d);

sort(queue);

if(d==0){

move=(head-l)+(queue[n-1]-l);

}

else{

move=(u-head)+(u-queue[0]);

}

return move;

}

int look(){

int move=0;

int d;

printf("enter the direction:(1->UP and 0->down):\n");

scanf("%d",&d);

sort(queue);

if(d==1){

move=(queue[n-1]-head)+(queue[n-1]-queue[0]);

}

else{

move=(head-queue[0])+(queue[n-1]-queue[0]);

}

return move;

}

int Clook(){

int move=0,d,u,count=0;

sort(queue);

printf("enter the direction:(1->UP and 0->down):\n");

scanf("%d",&d);

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

{

if(queue[i]>head){

u=i;

break;

}

}

if(d==1){

while(count!=n){

move+=abs(queue[u]-head);

head=queue[u];

u=(u+1)%n;

count++;

}

}

else{

u--;

while(count!=n){

if(u<0){

u=n-1;

}

move+=abs(queue[u]-head);

head=queue[u];

u=(u-1)%n;

count++;

}

}

return move;

}

int Cscan(){

int move=0;

int l,u;

printf("enter the lower and upper limit:\n");

scanf("%d%d",&l,&u);

move=Clook();

move+=2\*((u-queue[n-1])+(queue[0]-l));

return move;

}

int main(){

int ch;

printf("enter the number of the containers:\n");

scanf("%d",&n);

printf("enter the queue of addresses:\n");

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

{

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

}

printf("enter the head location:\n");

scanf("%d",&head);

do{

printf("1.FCFS\n2.SSTF\n3.SCAN\n4.LOOK\n5.C-LOOK\n6.C-SCAN\n7.EXIT\n");

scanf("%d",&ch);

switch(ch){

case 1:

printf("Disk Movement using fcfs Algorithm= %d",fcfs());

break;

case 2:

printf("Disk Movement using sstf Algorithm= %d",sstf());

break;

case 3:

printf("Disk Movement using scan Algorithm= %d",scan());

break;

case 4:

printf("Disk Movement using look Algorithm= %d",look());

break;

case 5:

printf("Disk Movement using C-look Algorithm= %d",Clook());

break;

case 6:

printf("Disk Movement using C-Scan Algorithm= %d",Cscan());

break;

case 7:

exit(0);

default:

printf("enter Valid choice!!!\n");

break;

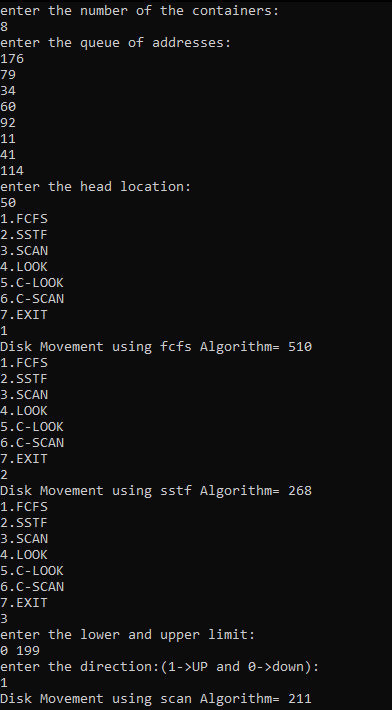
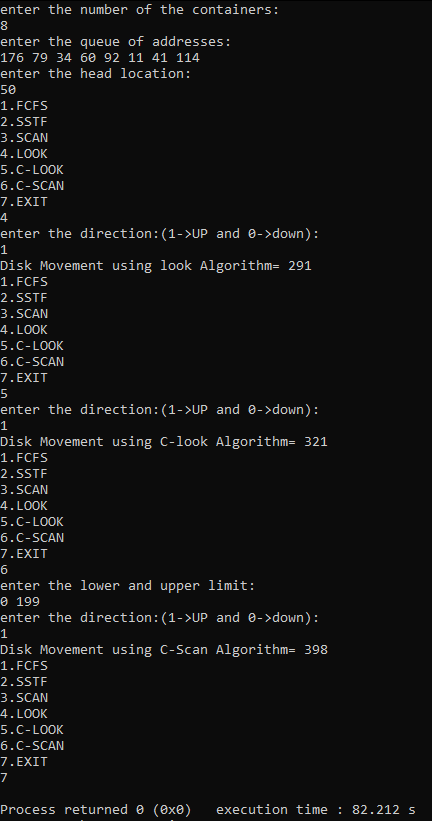
}

}while(ch!=7);

return 0;

}

**OUTPUT:**

****

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

**a) FIFO**

**b) LRU**

**c)Optimal**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_FRAMES 4

void printFrames(int frames[], int n) {

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

if (frames[i] == -1) {

printf("- ");

} else {

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

}

}

printf("\n");

}

int findLRUIndex(int counters[], int n) {

int minIndex = 0;

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

if (counters[i] < counters[minIndex]) {

minIndex = i;

}

}

return minIndex;

}

int findOptimalIndex(int pages[], int frames[], int n, int start) {

int index = -1;

int farthest = start;

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

int j;

for (j = start; j < n; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

index = i;

}

break;

}

}

if (j == n) {

return i;

}

}

return (index == -1) ? 0 : index;

}

void fifo(int pages[], int n) {

int frames[MAX\_FRAMES];

int frameIndex = 0;

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1;

}

printf("FIFO Page Replacement Algorithm:\n");

int pageFaults = 0;

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

int page = pages[i];

bool pageFound = false;

for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] == page) {

pageFound = true;

break;

}

}

if (!pageFound) {

frames[frameIndex] = page;

frameIndex = (frameIndex + 1) % MAX\_FRAMES;

pageFaults++;

}

printFrames(frames, MAX\_FRAMES);

}

printf("Total Page Faults: %d\n\n", pageFaults);

}

void lru(int pages[], int n) {

int frames[MAX\_FRAMES];

int counters[MAX\_FRAMES] = {0};

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1;

}

printf("LRU Page Replacement Algorithm:\n");

int pageFaults = 0;

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

int page = pages[i];

bool pageFound = false;

for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] == page) {

pageFound = true;

counters[j] = i;

break;

}

}

if (!pageFound) {

int lruIndex = findLRUIndex(counters, MAX\_FRAMES);

frames[lruIndex] = page;

counters[lruIndex] = i;

pageFaults++;

}

printFrames(frames, MAX\_FRAMES);

}

printf("Total Page Faults: %d\n\n", pageFaults);

}

void optimal(int pages[], int n) {

int frames[MAX\_FRAMES];

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1;

}

printf("Optimal Page Replacement Algorithm:\n");

int pageFaults = 0;

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

int page = pages[i];

bool pageFound = false;

for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] == page) {

pageFound = true;

break;

}

}

if (!pageFound) {

int optimalIndex = findOptimalIndex(pages, frames, n, i + 1);

frames[optimalIndex] = page;

pageFaults++;

}

printFrames(frames, MAX\_FRAMES);

}

printf("Total Page Faults: %d\n\n", pageFaults);

}

int main() {

int n;

printf("enter the number of pages:\n");

scanf("%d",&n);

int pages[n] ;

printf("enter the page indexes:\n");

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

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

}

int choice;

do {

printf("Page Replacement Algorithms:\n");

printf("1. FIFO\n");

printf("2. LRU\n");

printf("3. Optimal\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

fifo(pages, n);

break;

case 2:

lru(pages, n);

break;

case 3:

optimal(pages, n);

break;

case 4:

printf("Exiting the program.\n");

break;

default:

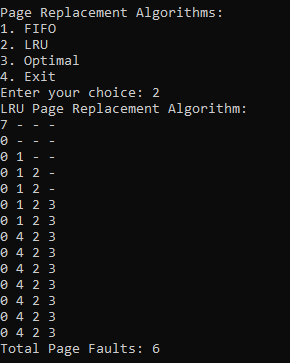
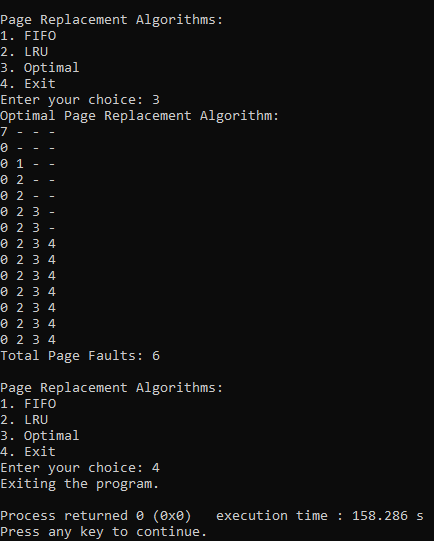
printf("Invalid choice. Please select a valid option.\n");

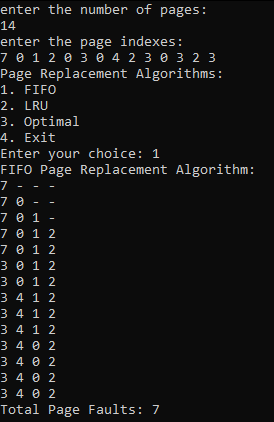
}

} while (choice != 4);

return 0;

}

**OUTPUT:**

****