JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

**OPERATING SYSTEM AND SYSTEM PROGRAMMING LAB**

**TOPIC: DISC SCHEDULING PROJECT**



Under the guidance of:

Ms. Anubhuti Roda Mohindra

|  |  |
| --- | --- |
| GROUP MEMBERS | ENROLLMENT |
| KHUSHI KALRA | **9920103025(F1)** |
| VAISHALI RANJAN | **9920103013(F1)** |
| AVIRAL GUPTA | **9920103021(F1)** |
| DEVANSH CHUGH | **9920103011(F1)** |
| ANYA RATHI | **9920103001(F1)** |

CODE:

#include<bits/stdc++.h>

#include <stdio.h>

#include<string.h>

#include <stdlib.h>

using namespace std;

#define HIGH 199

#define LOW 0

void fcfs();

void sstf();

void scan();

void cscan();

void look();

void clook();

int main(){

printf("Enter 1 for FCFS \nEnter 2 for SSTF \nEnter 3 for SCAN \nEnter 4 for CSCAN \nEnter 5 for LOOK \nEnter 6 for CLOOK \n");

int choice;

scanf("%d",&choice);

switch(choice)

{

case 1:

fcfs();

break;

case 2:

sstf();

break;

case 3:

scan();

break;

case 4:

cscan();

break;

case 5:

look();

break;

case 6:

clook();

break;

default:

printf("You fool, don't know english? Get Lost");

}

return 0;

}

void fcfs()

{

int queue[100], q\_size, head, seek =0, diff;

float avg;

printf("%s\n", "\*\*FCFS Disk Scheduling Algorithm\*\*");

printf("%s\n", "Enter the size of the queue");

scanf("%d", &q\_size);

printf("%s\n", "Enter queue elements");

for(int i=1; i<=q\_size; i++){

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

}

printf("%s\n","Enter initial head position");

scanf("%d", &head);

queue[0]=head;

for(int j=0; j<=q\_size-1; j++){

diff = abs(queue[j]-queue[j+1]);

seek += diff;

printf("Move from %d to %d with Seek %d\n",queue[j],queue[j+1],diff);

}

printf("\nTotal seek time is %d\t",seek);

avg = seek/(float)q\_size;

printf("\nAverage seek time is %f\t", avg);

}

void calculatedifference(int request[], int head,

int diff[][2], int n)

{

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

{

diff[i][0] = abs(head - request[i]);

}

}

// Find unaccessed track which is

// at minimum distance from head

int findMIN(int diff[][2], int n)

{

int index = -1;

int minimum = 1e9;

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

{

if (!diff[i][1] && minimum > diff[i][0])

{

minimum = diff[i][0];

index = i;

}

}

return index;

}

void sstf()

{

int n;

printf("%s\n", "Enter the size of the queue");

scanf("%d", &n);

int request[n];

printf("%s\n", "Enter queue elements");

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

{

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

}

int head;

printf("%s\n","Enter initial head position");

scanf("%d", &head);

if (n == 0)

{

return;

}

// Create array of objects of class node

int diff[n][2] = { { 0, 0 } };

// Count total number of seek operation

int seekcount = 0;

// Stores sequence in which disk access is done

int seeksequence[n + 1] = {0};

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

{

seeksequence[i] = head;

calculatedifference(request, head, diff, n);

int index = findMIN(diff, n);

diff[index][1] = 1;

// Increase the total count

seekcount += diff[index][0];

// Accessed track is now new head

head = request[index];

}

seeksequence[n] = head;

cout << "Total number of seek operations = "

<< seekcount << endl;

cout << "Seek sequence is : " << "\n";

// Print the sequence

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

{

cout << seeksequence[i] << " -> ";

}

}

void scan()

{

int size;

printf("%s\n", "Enter the size of the queue");

scanf("%d", &size);

int arr[size];

printf("%s\n", "Enter queue elements");

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

{

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

}

int head;

printf("%s\n","Enter initial head position");

scanf("%d", &head);

string direction = "left";

int seek\_count = 0;

int distance, cur\_track;

vector<int> left, right;

vector<int> seek\_sequence;

// appending end values

// which has to be visited

// before reversing the direction

if (direction == "left")

left.push\_back(0);

else if (direction == "right")

right.push\_back(200 - 1);

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

if (arr[i] < head)

left.push\_back(arr[i]);

if (arr[i] > head)

right.push\_back(arr[i]);

}

// sorting left and right vectors

std::sort(left.begin(), left.end());

std::sort(right.begin(), right.end());

// run the while loop two times.

// one by one scanning right

// and left of the head

int run = 2;

while (run--) {

if (direction == "left") {

for (int i = left.size() - 1; i >= 0; i--) {

cur\_track = left[i];

// appending current track to seek sequence

seek\_sequence.push\_back(cur\_track);

// calculate absolute distance

distance = abs(cur\_track - head);

// increase the total count

seek\_count += distance;

// accessed track is now the new head

head = cur\_track;

}

direction = "right";

}

else if (direction == "right") {

for (int i = 0; i < right.size(); i++) {

cur\_track = right[i];

// appending current track to seek sequence

seek\_sequence.push\_back(cur\_track);

// calculate absolute distance

distance = abs(cur\_track - head);

// increase the total count

seek\_count += distance;

// accessed track is now new head

head = cur\_track;

}

direction = "left";

}

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < seek\_sequence.size(); i++) {

cout << seek\_sequence[i] <<" -> ";

}

}

void cscan()

{

int queue[20], q\_size, head, i,j, seek=0, diff, max, temp, queue1[20], queue2[20], temp1=0, temp2=0;

float avg;

printf("%s\t", "Input no of disk locations");

scanf("%d", &q\_size);

printf("%s\t", "Enter initial head position");

scanf("%d", &head);

printf("%s\n","Enter disk positions to be read");

for(i=0; i<q\_size; i++) {

scanf("%d", &temp);

if(temp >= head){

queue1[temp1] = temp;

temp1++;

} else {

queue2[temp2] = temp;

temp2++;

}

}

//sort both arrays

for(i=0; i<temp1-1; i++){

for(j=i+1; j<temp1; j++){

if(queue1[i] > queue1[j]){

temp = queue1[i];

queue1[i] = queue1[j];

queue1[j] = temp;

}

}

}

for(i=0; i<temp2-1; i++){

for(j=i+1; j<temp2; j++){

if(queue2[i]>queue2[j]){

temp = queue2[i];

queue2[i] = queue2[j];

queue2[j] = temp;

}

}

}

//calculate closest edge

if(abs(head-LOW) >= abs(head-HIGH)){

for(i=1,j=0; j<temp1; i++,j++){

queue[i] = queue1[j];

}

queue[i] = HIGH;

queue[i+1] = 0;

for(i=temp1+3, j=0; j<temp2; i++, j++){

queue[i] = queue2[j];

}

} else {

for(i=1,j=temp2-1; j>=0; i++,j--){

queue[i] = queue2[j];

}

queue[i] = LOW;

queue[i+1] = HIGH;

for(i=temp2+3, j=temp1-1; j>=0; i++, j--){

queue[i] = queue1[j];

}

}

queue[0] = head;

for(j=0; j<=q\_size+1; j++){

diff=abs(queue[j+1] - queue[j]);

seek += diff;

printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

//seek = seek - max; //subtract seek time back to zero

printf("Total seek time is %d\n", seek);

avg = seek/(float)q\_size;

printf("Average seek time is %f\n", avg);

}

void look()

{

int queue[20], head, q\_size, i,j, seek=0, diff, max, temp, queue1[20], queue2[20], temp1=0, temp2=0;

float avg;

printf("%s\t", "Input the number of disk locations");

scanf("%d", &q\_size);

printf("%s\t", "Enter initial head position");

scanf("%d", &head);

printf("%s\n", "Enter disk positions to read");

for(i=0; i<q\_size; i++){

scanf("%d", &temp);

//queue1 - elems greater than head

if(temp >= head){

queue1[temp1] = temp;

temp1++;

} else {

queue2[temp2] = temp;

temp2++;

}

}

//sort queue1 - increasing order

for(i=0; i<temp1-1; i++){

for(j=i+1; j<temp1; j++){

if(queue1[i] > queue1[j]){

temp = queue1[i];

queue1[i] = queue1[j];

queue1[j] = temp;

}

}

}

//sort queue2 - decreasing order

for(i=0; i<temp2-1; i++){

for(j=i+1; j<temp2; j++){

if(queue2[i] < queue2[j]){

temp = queue2[i];

queue2[i] = queue2[j];

queue2[j] = temp;

}

}

}

if(abs(head-LOW) >= abs(head-HIGH)){

for(i=1,j=0; j<temp1; i++,j++){

queue[i] = queue1[j];

}

for(i=temp1+1, j=0; j<temp2; i++, j++){

queue[i] = queue2[j];

}

} else {

for(i=1,j=0; j<temp2; i++,j++){

queue[i] = queue2[j];

}

for(i=temp2+1, j=0; j<temp1; i++, j++){

queue[i] = queue1[j];

}

}

queue[0] = head;

for(j=0; j<q\_size; j++){

diff=abs(queue[j+1] - queue[j]);

seek += diff;

printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

printf("Total seek time is %d\n", seek);

avg = seek/(float)q\_size;

printf("Average seek time is %f\n", avg);

}

void clook()

{

int size;

printf("%s\n", "Enter the size of the queue");

scanf("%d", &size);

int arr[size];

printf("%s\n", "Enter queue elements");

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

{

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

}

int head;

printf("%s\n","Enter initial head position");

scanf("%d", &head);

int seek\_count = 0;

int distance, cur\_track;

vector<int> left, right;

vector<int> seek\_sequence;

// Tracks on the left of the

// head will be serviced when

// once the head comes back

// to the beginning (left end)

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

if (arr[i] < head)

left.push\_back(arr[i]);

if (arr[i] > head)

right.push\_back(arr[i]);

}

// Sorting left and right vectors

std::sort(left.begin(), left.end());

std::sort(right.begin(), right.end());

// First service the requests

// on the right side of the

// head

for (int i = 0; i < right.size(); i++) {

cur\_track = right[i];

// Appending current track to seek sequence

seek\_sequence.push\_back(cur\_track);

// Calculate absolute distance

distance = abs(cur\_track - head);

// Increase the total count

seek\_count += distance;

// Accessed track is now new head

head = cur\_track;

}

// Once reached the right end

// jump to the last track that

// is needed to be serviced in

// left direction

seek\_count += abs(head - left[0]);

head = left[0];

// Now service the requests again

// which are left

for (int i = 0; i < left.size(); i++) {

cur\_track = left[i];

// Appending current track to seek sequence

seek\_sequence.push\_back(cur\_track);

// Calculate absolute distance

distance = abs(cur\_track - head);

// Increase the total count

seek\_count += distance;

// Accessed track is now the new head

head = cur\_track;

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < seek\_sequence.size(); i++) {

cout << seek\_sequence[i] <<" -> ";

}

}