**lab 9**

Q. **Suppose the head of a moving- head disk with 200 tracks,**

**numbered 0 to 199 is currently serving request at tracks 143 and**

**has finished a request at track 125. The queue it requests is kept**

**in the FIFO order 86, 147, 91, 177, 94, 150, 102, 175, 130. Write**

**a program to calculate the total head movement using following**

**algorithms.**

• **FCFS**

**• SSTF**

**• SCAN**

**• C-SCAN**

**• LOOK**

**• C-LOOK**

**Code**

#include <iostream>

#include <cmath> // For abs()

#include <algorithm> // For sort(), but custom sorting is done here

using namespace std;

// Function to calculate FCFS head movement

int calculateFCFS(int requests[], int numRequests, int initialHead) {

int head = initialHead;

int totalMovement = 0;

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

totalMovement += abs(requests[i] - head);

head = requests[i];

}

return totalMovement;

}

// Function to calculate SSTF head movement

int calculateSSTF(int requests[], int numRequests, int initialHead) {

int head = initialHead;

int totalMovement = 0;

bool served[numRequests] = {0}; // Array to keep track of served requests

for (int count = 0; count < numRequests; ++count) {

int minDistance = 1e9; // Set to a large value

int closestIndex = -1;

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

if (!served[i]) {

int distance = abs(requests[i] - head);

if (distance < minDistance) {

minDistance = distance;

closestIndex = i;

}

}

}

totalMovement += minDistance;

head = requests[closestIndex];

served[closestIndex] = true;

}

return totalMovement;

}

// Function to calculate SCAN head movement

int calculateSCAN(int requests[], int numRequests, int initialHead, int diskSize) {

int sortedRequests[20];

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

sortedRequests[i] = requests[i];

}

// Sort requests in ascending order

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

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

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

int temp = sortedRequests[i];

sortedRequests[i] = sortedRequests[j];

sortedRequests[j] = temp;

}

}

}

int head = initialHead;

int totalMovement = 0;

// If head starts below the smallest request, move right first, then left

bool directionUp = head < sortedRequests[0];

if (directionUp) {

// Move right

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

if (sortedRequests[i] >= head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

// Move left

for (int i = numRequests - 1; i >= 0; --i) {

if (sortedRequests[i] < head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

} else {

// Move left

for (int i = numRequests - 1; i >= 0; --i) {

if (sortedRequests[i] <= head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

// Move right

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

if (sortedRequests[i] > head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

}

return totalMovement;

}

// Function to calculate C-SCAN head movement

int calculateCSCAN(int requests[], int numRequests, int initialHead, int diskSize) {

int sortedRequests[20];

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

sortedRequests[i] = requests[i];

}

// Sort requests in ascending order

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

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

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

int temp = sortedRequests[i];

sortedRequests[i] = sortedRequests[j];

sortedRequests[j] = temp;

}

}

}

int head = initialHead;

int totalMovement = 0;

// Move right

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

if (sortedRequests[i] >= head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

// Jump to the beginning of the disk

if (head < diskSize - 1) {

totalMovement += (diskSize - 1 - head);

head = 0;

}

// Move from the beginning of the disk

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

if (sortedRequests[i] < head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

return totalMovement;

}

// Function to calculate LOOK head movement

int calculateLOOK(int requests[], int numRequests, int initialHead) {

int sortedRequests[20];

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

sortedRequests[i] = requests[i];

}

// Sort requests in ascending order

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

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

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

int temp = sortedRequests[i];

sortedRequests[i] = sortedRequests[j];

sortedRequests[j] = temp;

}

}

}

int head = initialHead;

int totalMovement = 0;

// If head starts below the smallest request, move right first, then left

bool directionUp = head < sortedRequests[0];

if (directionUp) {

// Move right

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

if (sortedRequests[i] >= head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

// Move left

for (int i = numRequests - 1; i >= 0; --i) {

if (sortedRequests[i] < head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

} else {

// Move left

for (int i = numRequests - 1; i >= 0; --i) {

if (sortedRequests[i] <= head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

// Move right

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

if (sortedRequests[i] > head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

}

return totalMovement;

}

// Function to calculate C-LOOK head movement

int calculateCLOOK(int requests[], int numRequests, int initialHead) {

int sortedRequests[20];

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

sortedRequests[i] = requests[i];

}

// Sort requests in ascending order

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

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

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

int temp = sortedRequests[i];

sortedRequests[i] = sortedRequests[j];

sortedRequests[j] = temp;

}

}

}

int head = initialHead;

int totalMovement = 0;

// Move right

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

if (sortedRequests[i] >= head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

// Wrap around to the beginning and continue moving right

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

if (sortedRequests[i] < head) {

totalMovement += abs(sortedRequests[i] - head);

head = sortedRequests[i];

}

}

return totalMovement;

}

int main() {

int initialHead = 143;

int diskSize = 200; // Total number of tracks

int requests[] = {86, 147, 91, 177, 94, 150, 102, 175, 130};

int numRequests = sizeof(requests) / sizeof(requests[0]);

int choice;

while (true) {

// Menu for user input

cout << "\nDisk Scheduling Algorithms\n";

cout << "1. FCFS\n";

cout << "2. SSTF\n";

cout << "3. SCAN\n";

cout << "4. C-SCAN\n";

cout << "5. LOOK\n";

cout << "6. C-LOOK\n";

cout << "7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

int totalMovement = 0;

switch(choice) {

case 1:

totalMovement = calculateFCFS(requests, numRequests, initialHead);

cout << "Total head movement using FCFS: " << totalMovement << " tracks\n";

break;

case 2:

totalMovement = calculateSSTF(requests, numRequests, initialHead);

cout << "Total head movement using SSTF: " << totalMovement << " tracks\n";

break;

case 3:

totalMovement = calculateSCAN(requests, numRequests, initialHead, diskSize);

cout << "Total head movement using SCAN: " << totalMovement << " tracks\n";

break;

case 4:

totalMovement = calculateCSCAN(requests, numRequests, initialHead, diskSize);

cout << "Total head movement using C-SCAN: " << totalMovement << " tracks\n";

break;

case 5:

totalMovement = calculateLOOK(requests, numRequests, initialHead);

cout << "Total head movement using LOOK: " << totalMovement << " tracks\n";

break;

case 6:

totalMovement = calculateCLOOK(requests, numRequests, initialHead);

cout << "Total head movement using C-LOOK: " << totalMovement << " tracks\n";

break;

case 7:

cout << "Exiting program...\n";

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

**Output:**





