#### Disk Scheduling Algorithms

**Aim:**

To write a menu driven program to implement the following disk scheduling algorithms:

(a) FCFS (b) SCAN (c) C-SCAN

**Description:**

**First-Come, First-Served (FCFS)**

The First-Come, First-Served (FCFS) disk scheduling algorithm is the simplest form of disk scheduling. In FCFS, requests are processed in the order they arrive, regardless of their position on the disk. This method maintains a straightforward queue where each incoming request is appended to the end. When a request is completed, the next request in the queue is serviced. This algorithm is easy to implement and ensures fairness since every request is treated equally, and no request is prioritized over another. However, FCFS can lead to suboptimal performance, particularly when there are large variations in the location of the requested data. The disk arm might need to move back and forth significantly between requests, leading to high seek times and overall inefficiency. This effect, known as the "convoy effect," can cause the average wait time to increase dramatically, particularly in cases where a sequence of requests happens to require extensive movement of the disk arm.

**Algorithm:**

**Step 1:** Start

**Step 2:** Initialize the total head movement (TotalHeadMovement) to 0.

**Step 3:** Begin iterating through each request in the queue (rq[]).

**Step 4:** For each request:

* Calculate the head movement by taking the absolute difference between the current head position (initial\_pos) and the position of the request (rq[i]).
* Add this movement to TotalHeadMovement.
* Update order[] with the request order processed.
* Update initial\_pos to the current request position (rq[i]).

**Step 5:** After processing all requests, print the seek sequence (order[]) and the total head movement (TotalHeadMovement).

**Step 6:** Stop

**SCAN Algorithm**

The SCAN algorithm, also known as the Elevator Algorithm, improves upon FCFS by reducing the overall seek time through a more systematic approach. In SCAN, the disk arm moves in one direction, servicing all requests until it reaches the end of the disk. After reaching the end, the direction reverses, and the arm services requests on its way back. This back-and-forth movement is analogous to an elevator picking up passengers as it moves up and down a building. By continuously scanning the disk in both directions, the SCAN algorithm reduces the average seek time compared to FCFS. Requests are handled more efficiently because the arm movement is more predictable, and the maximum distance the arm travels between requests is minimized. However, one drawback is that SCAN can lead to starvation of requests that are consistently arriving at the end of the current scan direction, as they may have to wait until the disk arm reverses direction.

**Algorithm:**

**Step 1:** Start

**Step 2:** Initialize move to determine the direction of head movement (1 for high, 0 for low).

**Step 3:** Insert the boundary requests (size) into the rq[] array.

**Step 4:** Sort the requests (rq[]) to arrange them in ascending order.

**Step 5:** Find the position (index) where the head is currently located in the sorted rq[].

**Step 6:** If move is 1 (high direction):

* Move the head towards the higher requests first:
  + Calculate the head movement for each request from index to the end of rq[].
  + Update TotalHeadMovement and order[] accordingly.
* After reaching the end, move the head back to the lower boundary (size-1).
* Process requests from the beginning of rq[] up to index-1.
* Update TotalHeadMovement and order[] for each request.

**Step 7:** If move is 0 (low direction):

* Move the head towards the lower requests first:
  + Calculate the head movement for each request from index-1 to the beginning of rq[].
  + Update TotalHeadMovement and order[].
* After reaching the beginning, move the head to the upper boundary (size-1).
* Process requests from index to the end of rq[].
* Update TotalHeadMovement and order[] for each request.

**Step 8:** Display the seek sequence (order[]) and the total head movement (TotalHeadMovement).

**Step 9:** Stop

**Circular SCAN (C-SCAN)**

The Circular SCAN (C-SCAN) algorithm further refines the SCAN approach by treating the disk as a circular list. In C-SCAN, the disk arm moves in one direction, servicing all requests until it reaches the end of the disk. However, instead of reversing direction, the arm jumps back to the beginning of the disk and starts servicing requests in the same direction again. This method ensures a more uniform wait time distribution and avoids the potential starvation issue present in the standard SCAN algorithm. C-SCAN provides a more uniform service time because each request is guaranteed to be serviced within one complete pass of the disk arm. By not reversing direction and instead jumping back to the start, C-SCAN ensures that all requests are treated more equally over time, reducing the variability in wait times. This makes C-SCAN particularly well-suited for systems where requests are continuously arriving, as it balances the workload more evenly across the entire disk.

Algorithm:

**Step 1:** Start

**Step 2:** Initialize move to determine the direction of head movement (1 for high, 0 for low).

**Step 3:** Insert the boundary requests (size) into the rq[] array.

**Step 4:** Sort the requests (rq[]) to arrange them in ascending order.

**Step 5:** Find the position (index) where the head is currently located in the sorted rq[].

**Step 6:** If move is 1 (high direction):

* Move the head towards the higher requests first:
  + Calculate the head movement for each request from index to the end of rq[].
  + Update TotalHeadMovement and order[].
* After reaching the end, move the head to the lower boundary (0).
* Process requests from the beginning of rq[] up to index-1.
* Update TotalHeadMovement and order[] for each request.

**Step 7:** If move is 0 (low direction):

* Move the head towards the lower requests first:
  + Calculate the head movement for each request from index-1 to the beginning of rq[].
  + Update TotalHeadMovement and order[].
* After reaching the beginning, move the head to the upper boundary (size-1).
* Process requests from index to the end of rq[].
* Update TotalHeadMovement and order[] for each request.

**Step 8:** Display the seek sequence (order[]) and the total head movement (TotalHeadMovement).

**Step 9:** Stop

**Code:**

#include<stdio.h>

#include<stdlib.h>

void display\_out(int TotalHeadMovement,int order[], int n );

void fcfs\_sch(int rq[],int n,int initial\_pos,int TotalHeadMovement,int order[]);

void scan\_sch(int rq[],int n,int initial\_pos,int TotalHeadMovement,int order[]);

void c\_scan\_sch(int rq[],int n,int initial\_pos,int TotalHeadMovement,int order[]);

int size;

void main(){

    int rq[100],rq\_temp[100],i,j,n,initial\_pos;

    int choice;

    int TotalHeadMovement=0;

    int order[100];

    printf("Enter total disk size: ");

    scanf("%d",&size);

    printf("Enter the number of Requests: ");

    scanf("%d",&n);

    printf("Enter Request sequence: \n");

    for(i=0;i<n;i++){

        scanf("%d",&rq\_temp[i]);

    }

    printf("Enter initial head position: ");

    scanf("%d",&initial\_pos);

    printf("\n---------------------------------------------------------");

    printf("\nMenu\n1.FCFS\n2.SCAN\n3.C SCAN\n0.Exit\n");

    do{

        for(i=0;i<n;i++){

            rq[i]=rq\_temp[i];

        }

        printf("Enter choice: ");

        scanf("%d",&choice);

        switch(choice){

            case 1: fcfs\_sch(rq,n,initial\_pos,TotalHeadMovement,order);

                    break;

            case 2:scan\_sch(rq,n,initial\_pos,TotalHeadMovement,order);

                    break;

            case 3:c\_scan\_sch(rq,n,initial\_pos,TotalHeadMovement,order);

                    break;

            case 0:

                    break;

        }

    }while(choice!=0);

}

void fcfs\_sch(int rq[],int n,int initial\_pos,int TotalHeadMovement,int order[]){

    for(int i=0;i<n;i++){

        TotalHeadMovement+=abs(rq[i]-initial\_pos);

        order[i]=rq[i];

        initial\_pos=rq[i];

    }

    printf("FCFS Algorithm\n");

    display\_out(TotalHeadMovement,order,n);

}

void display\_out(int TotalHeadMovement,int order[], int n){

    printf("Seek sequence : ");

    for(int i=0;i<n;i++){

        printf("%d ",order[i]);

    }

    printf("\nTotalHeadMovement = %d \n\n",TotalHeadMovement);

}

void scan\_sch(int rq[],int n,int initial\_pos,int TotalHeadMovement,int order[])

{

    int move,i,j,k=0;

    int index;

    printf("SCAN Algorithm \nEnter direction to move(1= high and 0= low):  ");

    scanf("%d",&move);

    rq[n]=size;

    n++;

    //sorting rq

    for(i=0;i<n;i++){

        for(j=0;j<n-i-1;j++){

            if(rq[j]>rq[j+1]){

                int temp=rq[j];

                rq[j]=rq[j+1];

                rq[j+1]=temp;

            }

        }

    }

    printf("Sorted requests; ");

    for(i=0;i<n;i++){

        if(rq[i]!=size)

            printf("%d ",rq[i]);

    }

    printf("\n");

    for(i=0;i<n;i++){

        if(initial\_pos<rq[i]){

            index=i;

            break;

        }

    }

    if (move == 1){

        for(i=index;i<n;i++){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

        TotalHeadMovement+=abs(size-rq[i-1]-1);

        initial\_pos=size-1;

        for(i=index-1;i>=0;i--){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

    }

    else{

        for(i=index-1;i>=0;i--){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

        TotalHeadMovement+=abs(rq[i+1]);

        initial\_pos=0;

        for(i=index;i<n;i++){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

    }

    display\_out(TotalHeadMovement,order,k);

}

void c\_scan\_sch(int rq[],int n,int initial\_pos,int TotalHeadMovement,int order[])

{

    int move,i,j,k=0;

    int index;

    printf("C-SCAN Algorithm \nEnter direction to move(1=high and 0=low):  ");

    scanf("%d",&move);

    rq[n]=size;

    n++;

    for(i=0;i<n;i++){

        for(j=0;j<n-i-1;j++){

            if(rq[j]>rq[j+1]){

                int temp=rq[j];

                rq[j]=rq[j+1];

                rq[j+1]=temp;

            }

        }

    }

    printf("Sorted requests:");

    for(i=0;i<n;i++){

        printf("%d ",rq[i]);

    }

    printf("\n");

    for(i=0;i<n;i++){

        if(initial\_pos<rq[i]){

            index=i;

            break;

        }

    }

    if (move == 1){

        for(i=index;i<n;i++){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

        TotalHeadMovement+=abs(size-rq[i-1]-1);

        TotalHeadMovement+=abs(size-1-0);

        initial\_pos=0;

        for(i=0;i<=index-1;i++){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

    }

    else{

        for(i=index-1;i>=0;i--){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

        TotalHeadMovement+=abs(rq[i+1]);

        TotalHeadMovement+=abs(size-1-0);

        initial\_pos=size-1;

        for(i=n-1;i>=index;i--){

            TotalHeadMovement+=abs(rq[i] - initial\_pos);

            initial\_pos=rq[i];

            order[k++]=rq[i];

        }

    }

    display\_out(TotalHeadMovement,order,k);

}

**Output:**

PS D:\Coding\repositories\S4-OSlab> gcc .\disk\_scheduling.c

PS D:\Coding\repositories\S4-OSlab> ./a.exe

Enter total disk size: 100

Enter the number of Requests: 7

Enter Request sequence:

12 98 42 21 67 33 85

Enter initial head position: 7

Menu

1.FCFS

2.SCAN

3.C SCAN

0.Exit

Enter choice: 1

FCFS Algorithm

Seek sequence : 12 98 42 21 67 33 85

TotalHeadMovement = 300

Enter choice: 2

SCAN Algorithm

Enter direction to move(1= high and 0= low): 0

Sorted requests; 12 21 33 42 67 85 98

Seek sequence : 12 21 33 42 67 85 98 100

TotalHeadMovement = 112

Enter choice: 2

SCAN Algorithm

Enter direction to move(1= high and 0= low): 1

Sorted requests; 12 21 33 42 67 85 98

Seek sequence : 12 21 33 42 67 85 98 100

TotalHeadMovement = 94

Enter choice: 3

C-SCAN Algorithm

Enter direction to move(1=high and 0=low): 0

Sorted requests:12 21 33 42 67 85 98 100

Seek sequence : 100 98 85 67 42 33 21 12

TotalHeadMovement = 200

Enter choice: 3

C-SCAN Algorithm

Enter direction to move(1=high and 0=low): 1

Sorted requests:12 21 33 42 67 85 98 100

Seek sequence : 12 21 33 42 67 85 98 100

TotalHeadMovement = 193

Enter choice: 0

Exitted

**Result:**

The program has been executed and output has been verified.