ASSIGNMENT No: 8

Title : Disk scheduling algorithms

AIM : To implement C programs for Disk scheduling algorithms SSTF SCAN C-LOOK

INTRODUCTION: In operating systems, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down. Disk Scheduling Algorithms are used to reduce the total seek time of any request.

The purpose of this material is to provide one with help on disk scheduling algorithms. Hopefully with this, one will be able to get a stronger grasp of what disk scheduling algorithms do.

TYPES OF DISK SCHEDULING ALGORITHMS: Although there are other algorithms that reduce the seek time of all requests, I will only concentrate on the following disk scheduling algorithms: First Come-First Serve (FCFS) ,Shortest Seek Time First (SSTF) ,Elevator (SCAN) Circular SCAN (C-SCAN) ,LOOK C-LOOK.

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>


// Function to find the index of the closest track to the current head position

int findClosestTrack(int tracks[], int n, int head) {

   int minDist = abs(tracks[0] - head);

   int index = 0;

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

      int dist = abs(tracks[i] - head);

      if (dist < minDist) {

         minDist = dist;

         index = i;

      }

   }

   return index;

}
```

```
// SSTF Disk Scheduling Algorithm
void sstf(int tracks[], int n, int head) {
   int totalSeekTime = 0;
   printf("SSTF Disk Scheduling:\n");
   for (int i = 0; i < n; i++) {
      int closestTrackIndex = findClosestTrack(tracks, n, head);
      int seek = abs(tracks[closestTrackIndex] - head);
      totalSeekTime += seek;
      printf("Move head from %d to %d (Seek %d)\n", head, tracks[closestTrackIndex],
seek);
      head = tracks[closestTrackIndex];
      tracks[closestTrackIndex] = -1; // Mark track as visited
   }
   printf("Total seek time: %d\n", totalSeekTime);
}

// SCAN Disk Scheduling Algorithm
void scan(int tracks[], int n, int head) {
   int totalSeekTime = 0;
   int direction = 1; // 1 for moving towards the higher track numbers
   printf("SCAN Disk Scheduling:\n");
   while (1) {
      if (direction == 1) {
         int closestTrackIndex = -1;
         int minDist = INT_MAX;
         for (int i = 0; i < n; i++) {
            if (tracks[i] >= 0 && tracks[i] > head && abs(tracks[i] - head) < minDist) {
               minDist = abs(tracks[i] - head);
               closestTrackIndex = i;
            }
```

```
        }
        if (closestTrackIndex == -1) {
            // If no tracks are found in the current direction, change direction
            direction = 0; // Move towards lower track numbers
            continue;
        }
        int seek = abs(tracks[closestTrackIndex] - head);
        totalSeekTime += seek;
        printf("Move head from %d to %d (Seek %d)\n", head, tracks[closestTrackIndex],
seek);
        head = tracks[closestTrackIndex];
        tracks[closestTrackIndex] = -1; // Mark track as visited
      } else {
        int closestTrackIndex = -1;
        int minDist = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (tracks[i] >= 0 && tracks[i] < head && abs(tracks[i] - head) < minDist) {
                minDist = abs(tracks[i] - head);
                closestTrackIndex = i;
            }
        }
        if (closestTrackIndex == -1) {
            // If no tracks are found in the current direction, change direction
            direction = 1; // Move towards higher track numbers
            continue;
        }
        int seek = abs(tracks[closestTrackIndex] - head);
        totalSeekTime += seek;
        printf("Move head from %d to %d (Seek %d)\n", head, tracks[closestTrackIndex],
seek);
```

```
            head = tracks[closestTrackIndex];
            tracks[closestTrackIndex] = -1; // Mark track as visited
        }
        int allVisited = 1;
        for (int i = 0; i < n; i++) {
            if (tracks[i] >= 0) {
                allVisited = 0;
                break;
            }
        }
        if (allVisited) break;
    }
    printf("Total seek time: %d\n", totalSeekTime);
}

// C-Look Disk Scheduling Algorithm
void clook(int tracks[], int n, int head) {
    int totalSeekTime = 0;
    printf("C-Look Disk Scheduling:\n");
    while (1) {
        int closestTrackIndex = -1;
        int minDist = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (tracks[i] >= 0 && tracks[i] > head && abs(tracks[i] - head) < minDist) {
                minDist = abs(tracks[i] - head);
                closestTrackIndex = i;
            }
        }
        if (closestTrackIndex == -1) {
```

```
            // If no tracks are found in the current direction, wrap around
            closestTrackIndex = 0;
            for (int i = 0; i < n; i++) {
                if (tracks[i] >= 0 && tracks[i] < head && tracks[i] > tracks[closestTrackIndex]) {
                    closestTrackIndex = i;
                }
            }
        }
        int seek = abs(tracks[closestTrackIndex] - head);
        totalSeekTime += seek;
        printf("Move head from %d to %d (Seek %d)\n", head, tracks[closestTrackIndex],
seek);
        head = tracks[closestTrackIndex];
        tracks[closestTrackIndex] = -1; // Mark track as visited
        int allVisited = 1;
        for (int i = 0; i < n; i++) {
            if (tracks[i] >= 0) {
                allVisited = 0;
                break;
            }
        }
        if (allVisited) break;
    }
    printf("Total seek time: %d\n", totalSeekTime);
}

int main() {
    int n, head, choice;
    printf("Enter the number of tracks: ");
    scanf("%d", &n);
```

```
int tracks[n];
printf("Enter the track positions:\n");
for (int i = 0; i < n; i++) {
   scanf("%d", &tracks[i]);
}
printf("Enter the initial head position: ");
scanf("%d", &head);
while (1) {
   printf("\nDisk Scheduling Algorithms Menu:\n");
   printf("1. SSTF Disk Scheduling\n");
   printf("2. SCAN Disk Scheduling\n");
   printf("3. C-Look Disk Scheduling\n");
   printf("4. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);
   switch (choice) {
      case 1:
         sstf(tracks, n, head);
         break;
      case 2:
         scan(tracks, n, head);
         break;
      case 3:
         clook(tracks, n, head);
         break;
      case 4:
         exit(0);
      default:
         printf("Invalid choice. Please select a valid option.\n");
```

```
            break;
        }
    }
    return 0;
}
```

OUTPUT: