OS EXPERIMENT 10

Name: Mohammed Arhaan Ali
Div: S11
Roll No: 02

Aim: Write a program in C to do disk scheduling - FCFS, SCAN, C-SCAN

Theory:
File management and I/O (Input/Output) management are crucial components of operating systems responsible for handling the storage and retrieval of data from disk storage devices efficiently. Disk scheduling algorithms play a vital role in optimizing I/O operations by determining the order in which disk requests are serviced. Among the various disk scheduling algorithms, First-Come, First-Served (FCFS), SCAN, and C-SCAN are widely used. Let's delve into these concepts and understand how each algorithm works and its implications in disk management.

File Management: File management involves organizing and managing files on disk storage devices to facilitate efficient storage, retrieval, and manipulation of data. It encompasses various operations such as file creation, deletion, access control, and directory management. A file system is responsible for managing these operations and maintaining the structure and integrity of files and directories.

I/O Management: I/O management deals with managing Input/Output operations between the CPU, memory, and I/O devices such as disks, printers, and network interfaces. Disk I/O operations are particularly significant as they involve relatively slow mechanical devices compared to the CPU and memory. Disk scheduling algorithms are employed to optimize the order in which disk requests are serviced to minimize seek time and maximize disk throughput.

Disk Scheduling Algorithms:

1. First-Come, First-Served (FCFS): FCFS is the simplest disk scheduling algorithm, where disk requests are serviced in the order they arrive. It operates on the principle of fairness, as requests are processed based on their arrival times without any consideration for their locations on the disk.

Mechanism: When a disk request arrives, it is added to the end of the request queue. The disk scheduler services requests in the order they are queued. Each request is processed sequentially, starting from the innermost track to the outermost track of the disk. Implications: FCFS is easy to implement and ensures fairness in servicing requests. However, it may lead to longer seek times, especially if requests are scattered across the disk, resulting in poor disk performance.

2. SCAN (Elevator) Algorithm: SCAN, also known as the elevator algorithm, simulates the movement of an elevator moving up and down a building. It services requests in one direction until reaching the end of the disk, then reverses direction and continues servicing requests in the opposite direction.

Mechanism: The disk head starts from one end of the disk and moves towards the other end while servicing requests along its path. When it reaches the end of the disk, it reverses direction and starts moving towards the opposite end, servicing requests along the way. SCAN prevents the disk head from unnecessarily traversing the entire disk by changing direction at the disk boundaries. Implications: SCAN reduces the average seek time by prioritizing requests closer to the current position of the disk head. However, it may result in starvation for requests located at the extremes of the disk if there is a continuous stream of requests in one direction.

3. C-SCAN (Circular SCAN) Algorithm: C-SCAN is an enhancement of the SCAN algorithm that overcomes the potential starvation issue by treating the disk as a circular buffer. After reaching one end of the disk, the disk head jumps to the other end without servicing requests, ensuring fairness in request servicing.

Mechanism: Similar to SCAN, the disk head moves in one direction, servicing requests until reaching the end of the disk. Instead of reversing direction immediately, C-SCAN jumps to the other end of the disk without servicing requests. It then continues servicing requests in the same direction, preventing starvation for requests located at the disk boundaries. Implications: C-SCAN provides fairness in request servicing by preventing starvation for requests at the extremes of the disk. However, it may result in slightly higher average seek times compared to SCAN due to the jump between disk ends.

Program:

```
#include <stdio.h>

#include <stdlib.h>
#define MAX_REQUESTS 100 // Maximum number of disk requests
#define MAX_CYLINDERS 200 // Maximum number of cylinders

// Function prototypes
void fcfs(int requests[], int n, int initial_position);
void scan(int requests[], int n, int initial_position, int cylinders);
void c_scan(int requests[], int n, int initial_position, int cylinders);
int main() {
 int requests[MAX_REQUESTS];
 int n, initial_position, cylinders;
 printf("Enter the number of disk requests: ");
 scanf("%d", &n);
 printf("Enter the disk requests: ");
 for (int i = 0; i < n; i++) {
 scanf("%d", &requests[i]);
 }
 printf("Enter the initial position of the disk head: ");
 scanf("%d", &initial_position);

 printf("Enter the total number of cylinders on the disk: ");
 scanf("%d", &cylinders);
 printf("\nFirst-Come, First-Served (FCFS):\n");
 fcfs(requests, n, initial_position);
```

```
 printf("\nSCAN:\n");
 scan(requests, n, initial_position, cylinders);
 printf("\nC-SCAN:\n");
 c_scan(requests, n, initial_position, cylinders);
 return 0;
}
void fcfs(int requests[], int n, int initial_position) {
 int total_seek_time = 0;
 printf("Sequence of servicing requests:\n");
 for (int i = 0; i < n; i++) {
 printf("%d ", requests[i]);
 total_seek_time += abs(requests[i] - initial_position);
 initial_position = requests[i];
 }

 printf("\nTotal seek time: %d\n", total_seek_time);
}
void scan(int requests[], int n, int initial_position, int cylinders) {
 int total_seek_time = 0;
 int direction = 1; // 1 for right, -1 for left
 printf("Sequence of servicing requests:\n");
 // Sort requests to service in ascending order
 for (int i = 0; i < n - 1; i++) {
 for (int j = 0; j < n - i - 1; j++) {
 if (requests[j] > requests[j + 1]) {
 int temp = requests[j];
 requests[j] = requests[j + 1];
 requests[j + 1] = temp;
 }
 }
 }
 // Find the index where the disk head should reverse direction
 int reverse_index = 0;
 while (reverse_index < n && requests[reverse_index] <
initial_position) {

 reverse_index++;
```

```
}
// Service requests in one direction
for (int i = reverse_index - 1; i >= 0; i--) {
printf("%d ", requests[i]);
total_seek_time += abs(requests[i] - initial_position);
initial_position = requests[i];
}
// Service requests in the reverse direction
for (int i = reverse_index; i < n; i++) {
printf("%d ", requests[i]);
total_seek_time += abs(requests[i] - initial_position);
initial_position = requests[i];
}
printf("\nTotal seek time: %d\n", total_seek_time);
}
void c_scan(int requests[], int n, int initial_position, int cylinders) {
int total_seek_time = 0;
printf("Sequence of servicing requests:\n");

// Sort requests to service in ascending order
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (requests[j] > requests[j + 1]) {
int temp = requests[j];
requests[j] = requests[j + 1];
requests[j + 1] = temp;
}
}
}
// Service requests in one direction
for (int i = 0; i < n && requests[i] < initial_position; i++) {
printf("%d ", requests[i]);
total_seek_time += abs(requests[i] - initial_position);
initial_position = requests[i];
}
// Jump to the beginning of the disk
printf("%d ", 0);
```

```
 total_seek_time += initial_position;
 // Service requests in the reverse direction
 for (int i = n - 1; i >= 0 && requests[i] >= initial_position; i--) {
 printf("%d ", requests[i]);

 total_seek_time += abs(requests[i] - initial_position);
 initial_position = requests[i];
 }
 printf("\nTotal seek time: %d\n", total_seek_time);
}
```

Output:

```
PS C:\Users\arhaa\OneDrive\Desktop\OS> cd "c:\Users\arhaa\OneDrive\Desktop\
 } ; if ($?) { .\diskscheduling }
Enter the number of disk requests: 5
Enter the disk requests: 28 97 45 60 34
Enter the initial position of the disk head: 55
Enter the total number of cylinders on the disk: 200

First-Come, First-Served (FCFS):
Sequence of servicing requests:
28 97 45 60 34
Total seek time: 189

SCAN:
Sequence of servicing requests:
45 34 28 60 97
Total seek time: 96

C-SCAN:
Sequence of servicing requests:
28 0 97
Total seek time: 124
PS C:\Users\arhaa\OneDrive\Desktop\OS>
```

Conclusion: Thus we have successfully implemented disk scheduling algorithms - SCAN, C-SCAN,FCFS.