K. J. Somaiya School of Engineering, Mumbai-77
Somaiya Vidyavihar University
Department of Computer Engineering

TITLE: Implementation of Disk scheduling policies

AIM: Implementation of Disk scheduling algorithms - FCFS, SSTF, SCAN, CSCAN, LOOK, CLOOK (Any two) as instructed by instructor)

Expected Outcome of Experiment:

CO 4. To understand various Memory, I/O and File management techniques.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems", McGraw Hill Third Edition.**
3. **Sumitabha Das " UNIX Concepts & Applications", McGraw Hill Second Edition.**

Pre Lab/ Prior Concepts:

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.
Goal of Disk Scheduling Algorithm

FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

SSTF Scheduling Algorithm

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.
It allows the head to move to the closest track in the service queue.

Scan Algorithm

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns back and moves in the reverse direction satisfying requests coming in its path.
It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Look Scheduling

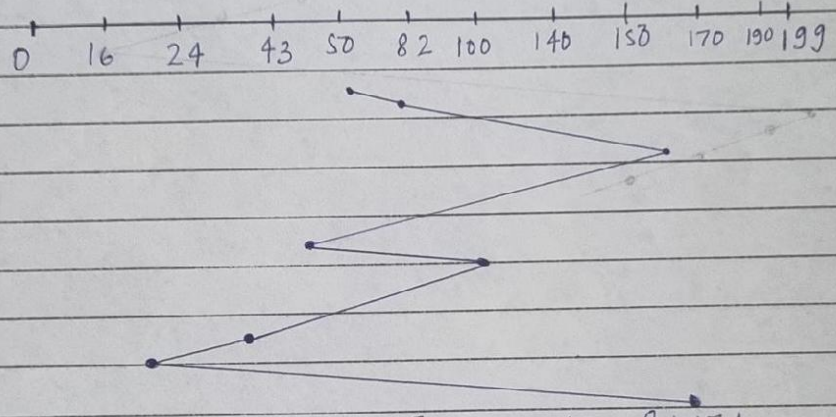
It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no

more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

Assigned Algorithm 1 details:

Disk Scheduling

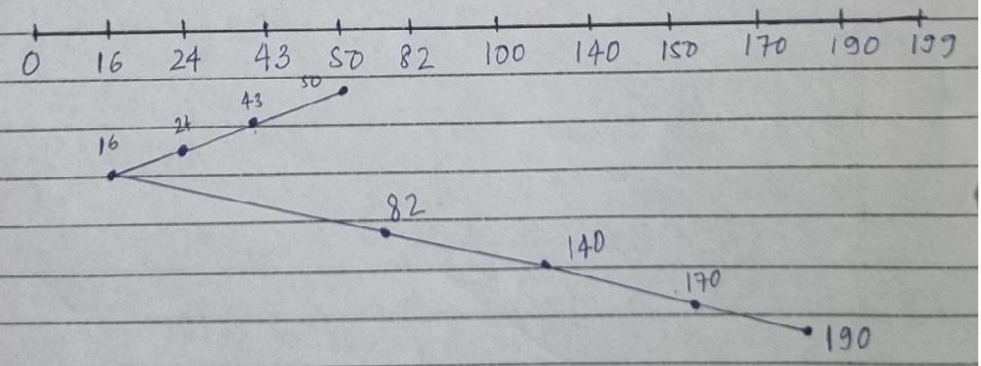
(1) FCFS (82, 170, 43, 140, 24, 16, 190)



Seek Time : 32 + 88 + 127 + 97 + 116 + 8 + 174
= 642 //

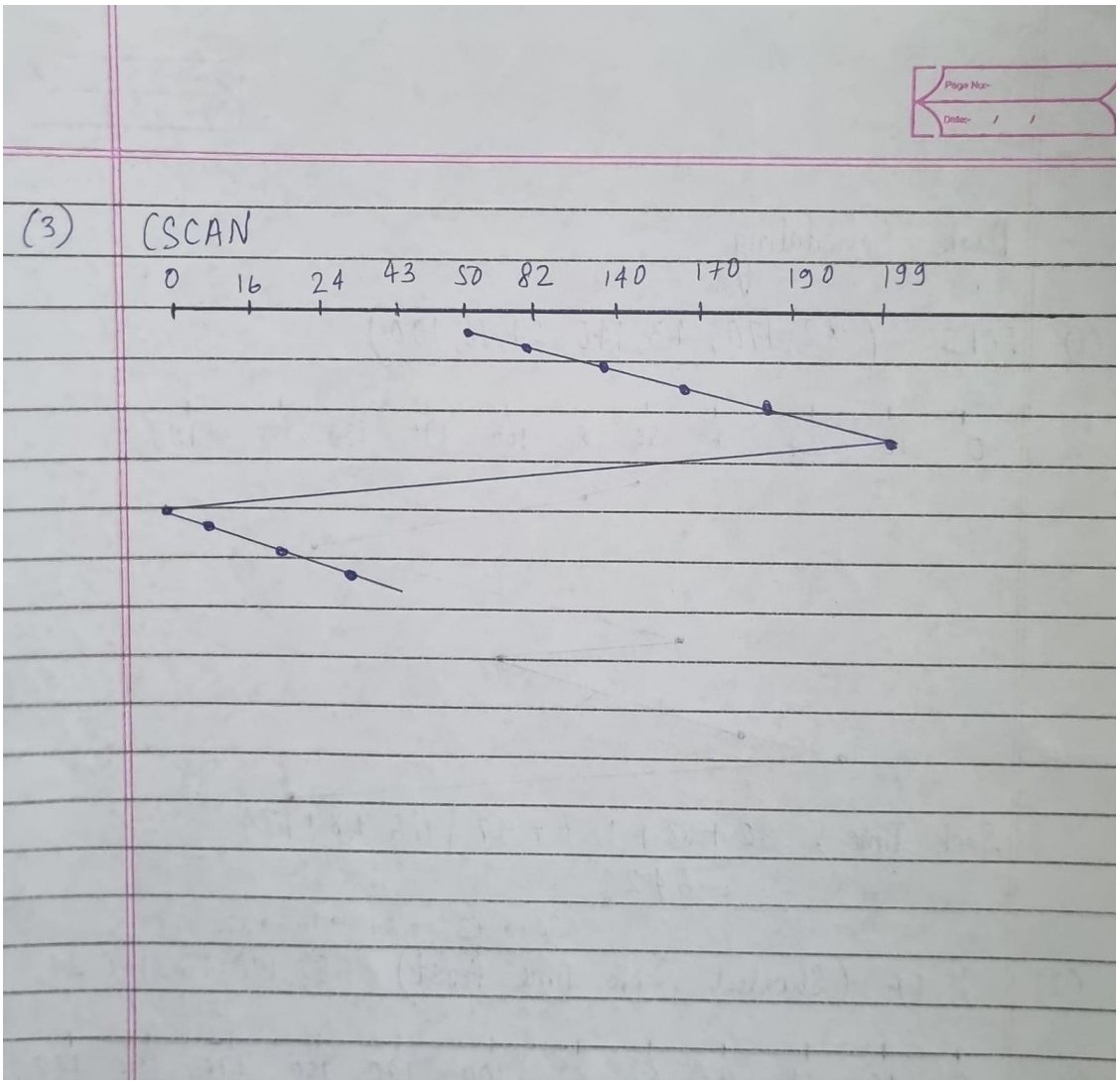
(2) SSTF (Shortest Seek Time first) (82, 170, 43, 140, 24, 16, 190)

50 → 43 → 24 → 16 → 82



Seek Time : 7 + 19 + 8 + 66 + 58 + 30 + 20
= 208.

Assigned Algorithm 2 details:



Source code:

FCFS:

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int main() {
    int current_position = 80;
    vector<int> requests = {67, 12, 15, 45, 48, 50, 109, 89, 56, 59, 34, 88, 113, 24, 22};
    int total_seek_time = 0;

    cout << "FCFS Seek Order: ";
    for (int req : requests) {
        int distance = abs(current_position - req);
        total_seek_time += distance;
        current_position = req;
        cout << req << " ";
    }

    cout << "\nTotal Seek Time (FCFS): " << total_seek_time << endl;
    return 0;
}
```

CSCAN:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    int current_position = 80;
    vector<int> requests = {67, 12, 15, 45, 48, 50, 109, 89, 56, 59, 34, 88, 113, 24, 22};
```

```
    const int DISK_START = 0;
    const int DISK_END = 199;

    vector<int> higher, lower;
    for (int req : requests) {
        if (req >= current_position)
            higher.push_back(req);
        else
            lower.push_back(req);
    }
    sort(higher.begin(), higher.end());
    sort(lower.begin(), lower.end());

    int total_seek_time = 0;
    total_seek_time += DISK_END - current_position;
    total_seek_time += DISK_END - DISK_START;
    if (!lower.empty()) {
        total_seek_time += lower.back() - DISK_START;
    }
    cout << "C-SCAN Seek Order: ";
    for (int h : higher) cout << h << " ";
    cout << DISK_END << " " << DISK_START << " ";
    for (int l : lower) cout << l << " ";

    cout << "\nTotal Seek Time (C-SCAN): " << total_seek_time << endl;

    return 0;
}
```

Output screenshots:

```
($?) { .\fcfsdiskscheduling }
FCFS Seek Order: 67 12 15 45 48 50 109 89 56 59 34 88 113 24 22
Total Seek Time (FCFS): 416
PS C:\Users\Satka\OneDrive\Desktop\AOA>
```

```
C-SCAN Seek Order: 88 89 109 113 199 0 12 15 22 24 34 45 48 50 56 59 67
Total Seek Time (C-SCAN): 385
PS C:\Users\Satka\OneDrive\Desktop\AOA>
```

Conclusion:

Learnt the code for FCFS and SCAN algorithms of disk scheduling.

Post Lab Descriptive Questions

i. Explain how SSTF can lead to starvation. Can this problem occur in SCAN or LOOK? Justify your answer.

ii. How does the size of the disk (number of tracks) influence the performance of the SCAN, C-SCAN, and LOOK algorithms? Provide examples to support your answer.

iii. Suppose that a disk drive has 200 cylinders numbered 0 to 199. Consider a disk queue with I/O requests on the following cylinders in their arriving order: 54, 95, 73, 130, 25, 48, 110, 30, 45, 180, and 190. The disk head is assumed to be at cylinder 24. Starting from the current head position, what is the total distance (in cylinders) the disk arm moves to satisfy all the pending requests for the SCAN and LOOK disk-scheduling algorithm?

Answers:

i)
- SSTF (Shortest Seek Time First) can lead to starvation because it prioritizes requests closest to the current head position, potentially leaving requests at the outer tracks unserved if a continuous stream of requests for inner tracks arrives.
- SSTF aims to minimize seek time by always servicing the request with the shortest distance from the current head position.
- If a series of requests arrive for tracks near the current position, the algorithm will repeatedly service those requests, effectively ignoring requests at the outer tracks.
- This can lead to a situation where requests at the outer tracks are indefinitely delayed or never serviced, resulting in starvation

While SCAN and LOOK algorithms also have limitations, they are designed to prevent starvation by moving the disk head in a single direction and servicing requests along the way.

- Both SCAN and LOOK algorithms are designed to prevent starvation by ensuring that all requests are eventually serviced.

• SCAN and LOOK algorithms, by moving the head in a single direction and servicing requests along the way, ensure that all requests are eventually processed, thus preventing starvation.

ii) As disk size (number of tracks) increases, the performance of SCAN, C-SCAN, and LOOK algorithms, which are disk scheduling algorithms, can be affected, primarily by increasing seek times and potentially waiting times.

1. SCAN (Elevator Algorithm)
   • Head moves to the end, then reverses.
   • Larger disk → longer travel distance, increasing seek and waiting times.
   • Example: For requests at 50, 200, 900 on a 1000-track disk, the head goes from 50 → 900 → back. On a 2000-track disk, it travels more, increasing delay.
2. C-SCAN
   • Head moves in one direction, then jumps to start.
   • Bigger disks → longer return jump, adding delay to low-track requests.
   • Example: 50 → 900 → 1000 → jump to 0 → 200. A 2000-track disk increases the jump time.
3. LOOK
   • Head moves only up to last request in direction.
   • More efficient on large disks than SCAN, but spread-out requests still increase seek time.
   • Example: 50 → 900 → 200 is faster than going to 1000, but still longer on bigger disks

iii. Suppose that a disk drive has 200 cylinders numbered 0 to 199. Consider a disk queue with I/O requests on the following cylinders in their arriving order: 54, 95, 73, 130, 25, 48, 110, 30, 45, 180, and 190. The disk head is assumed to be at cylinder 24. Starting from the current head position, what is the total distance (in cylinders) the disk arm moves to satisfy all the pending requests for the SCAN and LOOK disk-scheduling algorithm?

Date: _____