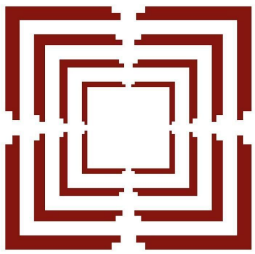


Ahmedabad
University

CSE524:Parallel and Distributed Systems

Faculty : Srikrishnan Divakaran

Group : 3 Project Report

Mechanisms and Policies for Designing Multi-head Disk Scheduling Algorithms

Name	Enrollment No.
Chirayu Vithalani	AU1940160
Darsh Patel	AU1940150
Devanshu Magiawala	AU1940190

Submitted on : 16th December 2021

Table of Contents

Multiple Head Disc Scheduling (MHDS) Algorithm

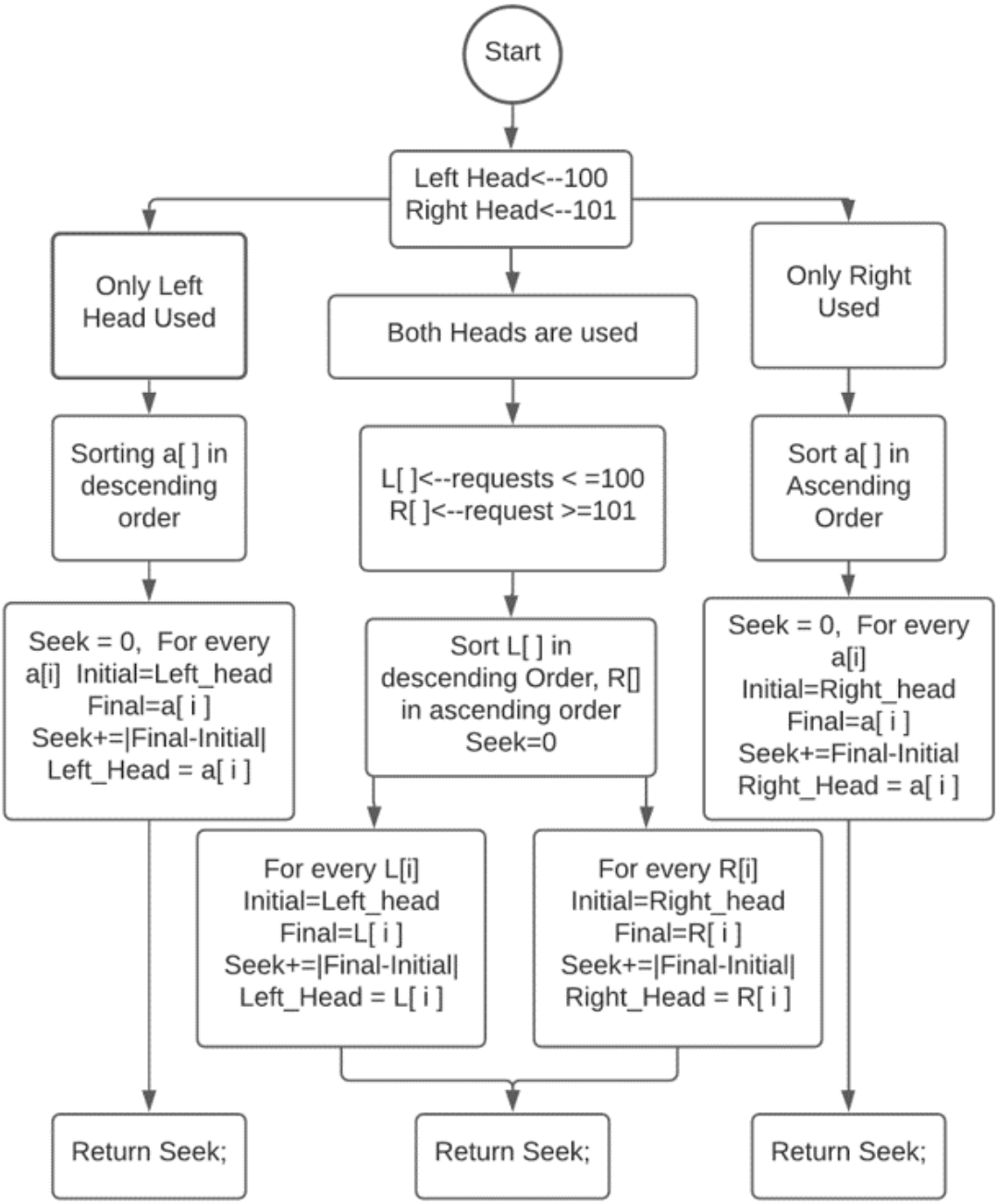
Assumptions

- Two heads will be used which will serve the I/O requests from the processor
- We will have the information on the number of requests and the order in which they arrive before we use the algorithm to schedule them
- Each I/O request will be served in unit time. The time required to serve each request will be the same.
- The disc has track numbers ranging from 1 to 200 both inclusive.
- The requests from the processor in the form of track numbers will be strictly between 1 to 200 both inclusive.

Mechanism of the Disk Scheduling Algorithm

1. The algorithm requests for the number of requests to be served and the order in which they were requested by the processor.
2. There are two HEADS that will serve these requests. One is termed Left Head and the other is termed Right Head.
3. Left Head and Right Head are initially at track numbers 100 and 101 respectively.
4. Left Head will serve the requests from 1 to 100 while Right Head will serve the requests with track numbers in the range 101 to 200.
5. The track numbers which need to be served are stored in an array.
6. The entire array will be scanned and if no request is found between 1 to 100, only the Right Head will be used.
7. The array will now be sorted in ascending order and the seek time can be calculated by the difference between the initial position of the Right Head and the final position of the Right Head every time the request is served.
8. The initial position of the Right Head is the track number it was previously at and the final position is the position where the Right Head is currently serving the track number. This step applies to each track number.
9. Similarly, if no request is found between 101 to 200, only the Left Head will be used.
10. The array will be sorted in decreasing order as the initial position of the Left Head is 100.
11. The seek time can be calculated by the absolute difference between the initial position of the Left Head and the final position of the Left Head every time the request is served.
12. The initial position of the Left Head is the track number it was previously at and the final position is the position where the Left Head is currently serving the track number. This step applies to each track number.
13. In the case when there are requests from both ranges 1 to 100 and 101 to 200, both the Heads will be used to serve the requests.
14. The requests which are in the range 1 to 100 will be stored in an array “L” and steps 7 and 8 will be applied in this new array to serve the requests and calculate the seek time for this array only.
15. The requests which are in the range 101 to 200 will be stored in an array “R” and steps 10 and 11 will be applied in this new array to serve the requests and calculate the seek time for this array only.
16. Finally, the seek time for both arrays will be added to get the seek time for the original array which contained the track numbers requested by the processor.[2]

The Process Flow



Code

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cout << "Please enter number of requests "
         << "\n";
    cin >> n;
    vector<int> v;
    cout << "Please enter the requests in range 1 to 200"
         << "\n";
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        v.push_back(x);
    }

    int leftHead = 100;
    int rightHead = 101;
    bool left = false;
    bool right = false;
    for (int i = 0; i < n; i++)
    {
        if (v[i] <= 100)
        {
            left = true;
        }
        else
        {
            right = true;
        }
    }

    if (right == false)
    {
        sort(v.rbegin(), v.rend());
        int seek = 0;

        for (int i = 0; i < n; i++)
        {
            leftHead -= v[i];
```

```
            seek += leftHead;
            cout << "Left Head is serving I/O request at track number ";
            cout << leftHead << "\n";
        }

        cout << "Total Head movement is ";
        cout << seek << "\n";
    }

    else if (left == false)
    {
        sort(v.begin(), v.end());
        int seek = 0;

        for (int i = 0; i < n; i++)
        {
            rightHead = v[i] - rightHead;
            seek += rightHead;
            cout << "Right Head is serving I/O request at track number ";
            cout << rightHead << "\n";
        }

        cout << "Total head movement is ";
        cout << seek << "\n";
    }

    else if (left == true && right == true)
    {

        int seek = 0;

        vector<int> l, r;
        for (int i = 0; i < n; i++)
        {
            if (v[i] <= 100)
            {

                l.push_back(v[i]);
            }
            else
            {
                r.push_back(v[i]);
            }
        }
```

```
        sort(l.rbegin(), l.rend());
        sort(r.begin(), r.end());

        int x = leftHead - l[0];
        int y = r[0] - rightHead;

        if (x <= y)
        {

            cout << "Left head will move first"
                 << "\n";

            for (int i = 0; i < l.size(); i++)
            {
                int initial = leftHead;
                int final = l[i];
                leftHead = l[i];
                cout << "Left Head is serving I/O request at track number ";
                cout << leftHead << "\n";
                seek += initial - final;
            }

            cout << "Now, Left Head will stop serving I/O requests"
                 << "\n";
            cout << "Right Head will start serving I/O requests from now on "
                 << "\n";

            for (int i = 0; i < r.size(); i++)
            {
                int initial = rightHead;
                int final = r[i];

                rightHead = r[i];
                cout << "Right Head is serving I/O request at track number ";
                cout << rightHead << "\n";

                seek += final - initial;
            }

            cout << "Total head movement is ";
            cout << seek;
        }

        else
        {
```

```
            cout << "Right Head will move first"
                 << "\n";

            for (int i = 0; i < r.size(); i++)
            {
                int initial = rightHead;
                int final = r[i];

                rightHead = r[i];
                cout << "Right Head is serving I/O request at track number ";
                cout << rightHead << "\n";

                seek += final - initial;
            }

            cout << "Now, Right Head will stop serving I/O requests"
                 << "\n";
            cout << "Left Head will start serving I/O requests from now on "
                 << "\n";

            for (int i = 0; i < l.size(); i++)
            {
                int initial = leftHead;
                int final = l[i];
                leftHead = l[i];
                cout << "Left Head is serving I/O request at track number ";
                cout << leftHead << "\n";
                seek += initial - final;
            }

            cout << "Total head movement is ";
            cout << seek;
        }
    }

    return 0;
}
```

Data Structures Used

- Only a single-dimensional array is used to store the track numbers where I/O is requested

Concurrency and Mutual Exclusion

- Each track number can be accessed for a read or write operation by only one Head.
- The Heads will cover a range that does not overlap with each other.
- Left Head will only serve requests from 1 to 100 whereas Right Head will only serve requests from 101 to 200.

Comparison with Standard Scheduling Algorithms

Performance of this proposed Multi-Head Disc Scheduling Algorithm against standard Disc scheduling algorithms like SSTF, CSCAN, FCFS

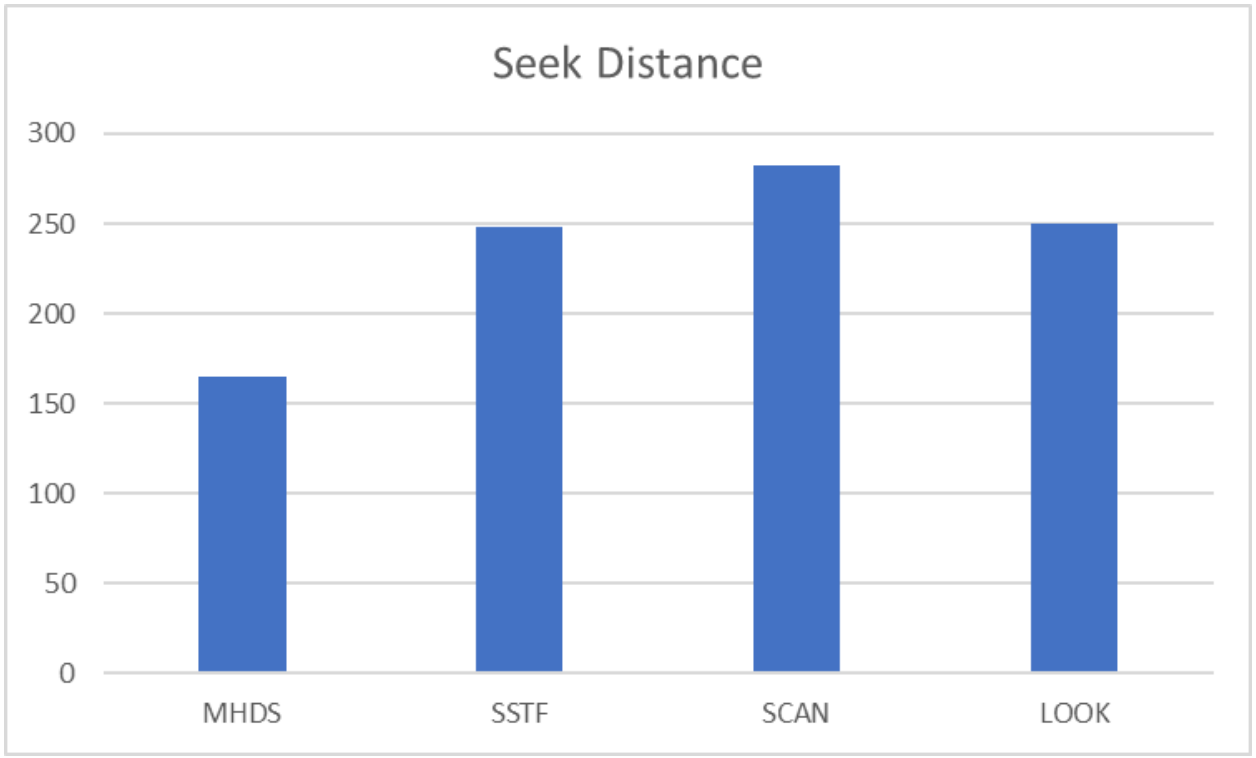
We will use the term MHDS for our proposed algorithm.

[1] Example 1: Suppose the order of requests for I/O operations is
55 58 39 18 90 160 150 38 184

```
Please enter number of requests
9
Please enter the requests in range 1 to 200
55 58 39 18 90 160 150 38 184
Left head will move first
Left Head is serving I/O request at track number 90
Left Head is serving I/O request at track number 58
Left Head is serving I/O request at track number 55
Left Head is serving I/O request at track number 39
Left Head is serving I/O request at track number 38
Left Head is serving I/O request at track number 18
Now, Left Head will stop serving I/O requests
Right Head will start serving I/O requests from now on
Right Head is serving I/O request at track number 150
Right Head is serving I/O request at track number 160
Right Head is serving I/O request at track number 184
Total head movement is 165
```

The initial head position for SSTF, SCAN and LOOK is 100

Scheduling Algorithm	Seek Distance
MHDS	165
SSTF	248
SCAN	282
LOOK	250

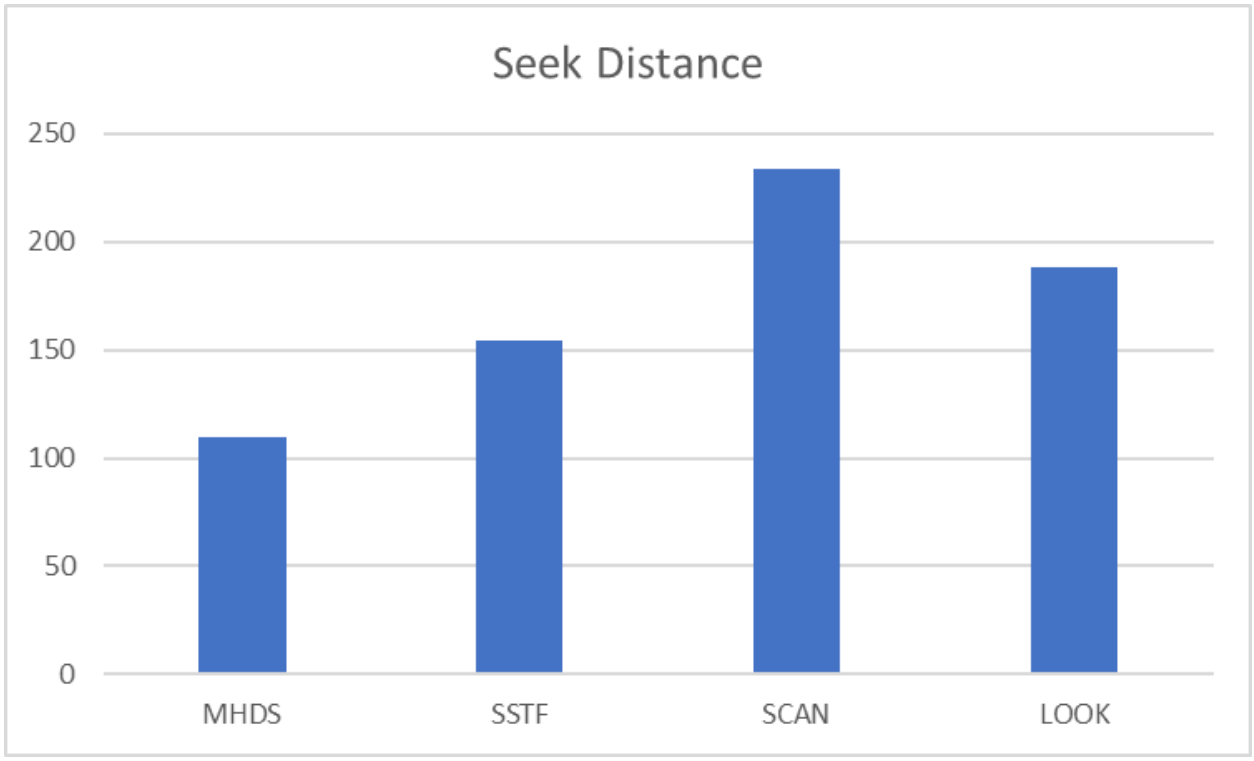


[1] Example 2: Suppose the order of requests for I/O operations is
86 147 66 177 94 150 102 175 130

```
Please enter number of requests
9
Please enter the requests in range 1 to 200
86 147 66 177 94 150 102 175 130
Right Head will move first
Right Head is serving I/O request at track number 102
Right Head is serving I/O request at track number 130
Right Head is serving I/O request at track number 147
Right Head is serving I/O request at track number 150
Right Head is serving I/O request at track number 175
Right Head is serving I/O request at track number 177
Now, Right Head will stop serving I/O requests
Left Head will start serving I/O requests from now on
Left Head is serving I/O request at track number 94
Left Head is serving I/O request at track number 86
Left Head is serving I/O request at track number 66
Total head movement is 110
```

The initial head position for SSTF, SCAN and LOOK is 100

Scheduling Algorithm	Seek Distance
MHDS	110
SSTF	154
SCAN	234
LOOK	188



References

[1] Singh, K., Rastogi, D., & Singh, D. (2015, February). Optimized Two Head Disk Scheduling Algorithm (OTHDSA). In *2015 Fifth International Conference on Advanced Computing & Communication Technologies* (pp. 234-240). IEEE.

[2] Parekh, H. B., Saksena, A., Ringshia, A., & Chaudhari, S. (2017, February). Simulation of a two head disk scheduling algorithm: An algorithm determining the algorithm to be imposed on the heads based on the nature of track requests. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)* (pp. 1-5). IEEE.