Week 10 – Assignments

1. Explain in detail about N STEP LOOK Algorithm with appropriate c program (A/P)

LOOK Disk Scheduling Algorithm

- Difficulty Level : Hard
- Last Updated : 06 Oct, 2021

- Read
- Discuss

Prerequisite: Disk Scheduling Algorithms
Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if *LOOK* disk scheduling algorithm is used. Also, write a program to find the seek sequence using *LOOK* disk scheduling algorithm.

LOOK Disk Scheduling Algorithm:
LOOK is the advanced version of SCAN (elevator) disk scheduling algorithm which gives slightly better seek time than any other algorithm in the hierarchy *(FCFS->SRTF->SCAN->C-SCAN->LOOK)*. The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also “looks” ahead as if there are more tracks that are needed to be serviced in the same direction. If there are no pending requests in the moving direction the head reverses the direction and start servicing requests in the opposite direction.
The main reason behind the better performance of LOOK algorithm in comparison to SCAN is because in this algorithm the head is not allowed to move till the end of the disk.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
2. The initial direction in which head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction head is moving.
4. The head continues to move in the same direction until all the request in this direction are finished.
5. While moving in this direction calculate the absolute distance of the track from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.

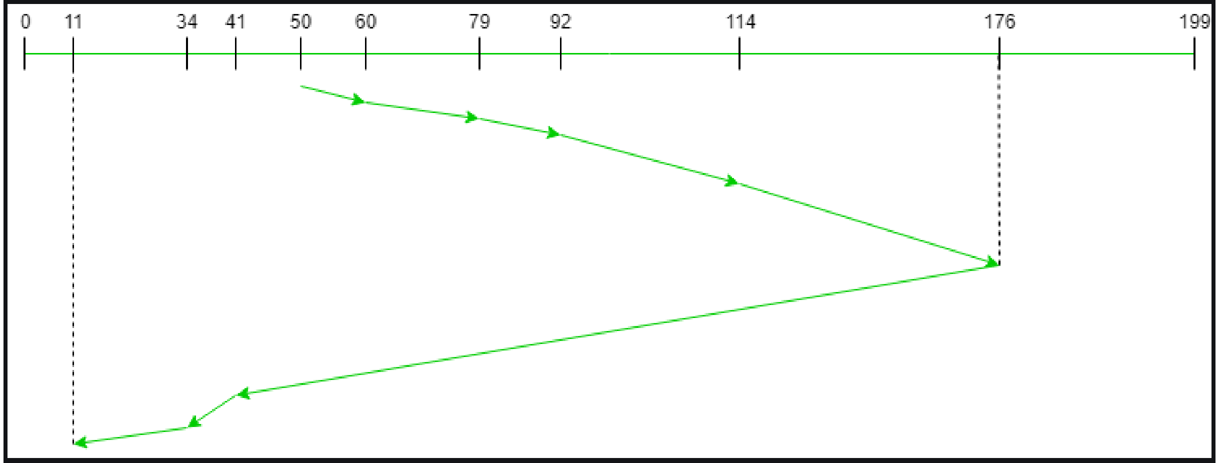
8. Go to step 5 until we reach at last request in this direction.
9. If we reach where no requests are needed to be serviced in this direction reverse the direction and go to step 3 until all tracks in request array have not been serviced.

Examples:

```
Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
Direction = right (We are moving from left to right)

Output:
Initial position of head: 50
Total number of seek operations = 291
Seek Sequence is
60
79
92
114
176
41
34
11
```

The following chart shows the sequence in which requested tracks are serviced using LOOK.



Therefore, the total seek count is calculated as:

```
= (60-50)+(79-60)+(92-79)
              +(114-92)+(176-114)
              +(176-41)+(41-34)+(34-11)
```

Implementation:
Implementation of LOOK algorithm is given below.
Note: The distance variable is used to store the absolute distance between the head and current track position. disk_size is the size of the disk. Vectors left and

right stores all the request tracks on the left-hand side and the right-hand side of the initial head position respectively.

- C++
- Java
- Python3
- C#
- Javascript

```
// C++ program to demonstrate
// LOOK Disk Scheduling algorithm
int size = 8;
#include <bits/stdc++.h>
using namespace std;

// Code by Vikram Chaurasia

int disk_size = 200;

void LOOK(int arr[], int head, string direction)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;

    // appending values which are
    // currently at left and right
    // direction from the head.
    for (int i = 0; i < size; i++) {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }

    // sorting left and right vectors
    // for servicing tracks in the
    // correct sequence.
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());

    // run the while loop two times.
    // one by one scanning right
    // and left side of the head
    int run = 2;
    while (run--) {
        if (direction == "left") {
            for (int i = left.size() - 1; i >= 0; i--) {
                cur_track = left[i];

                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);

                // calculate absolute distance
                distance = abs(cur_track - head);

                // increase the total count
                seek_count += distance;

                // accessed track is now the new head
                head = cur_track;
            }
            // reversing the direction
            direction = "right";
        }
        else if (direction == "right") {
            for (int i = 0; i < right.size(); i++) {
                cur_track = right[i];
                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);

                // calculate absolute distance
                distance = abs(cur_track - head);

                // increase the total count
                seek_count += distance;

                // accessed track is now new head
                head = cur_track;
            }
            // reversing the direction
            direction = "left";
        }
    }

    cout << "Total number of seek operations = "
         << seek_count << endl;

    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < seek_sequence.size(); i++) {
        cout << seek_sequence[i] << endl;
    }
}

// Driver code
int main()
{

    // request array
    int arr[size] = { 176, 79, 34, 60,
                      92, 11, 41, 114 };
    int head = 50;
    string direction = "right";

    cout << "Initial position of head: "
         << head << endl;

    LOOK(arr, head, direction);

    return 0;
}
```

Output:

```
Initial position of head: 50

Total number of seek operations = 291

Seek Sequence is

60

79

92

114

176

41

34

11
```

2. Explain in detail about F SCAN Algorithm with appropriate c program (A/P)

FScan disk scheduling algorithm

- Difficulty Level : Medium
- Last Updated : 11 Aug, 2020

- Read
- Discuss

Fixed period SCAN (FSCAN) disk scheduling algorithm mainly focuses on handling high variance in shortest seek time first (SSTF). SCAN algorithm is also proposed to handle above mentioned situation but using SCAN algorithm causes long delay while handling requests which are at extremes of disk. FSCAN algorithm determines how read and write head of disk will move in order to handle issue of handling issue of high variance of SSTF.

How it works?

FSCAN makes use of two queues, one of queues stores old r/w requests and other queue stores new r/w requests. When old requests are handled then only new requests are processed. Variations of FSCAN algorithm can also consist of N queues which in turn will make response time faster.

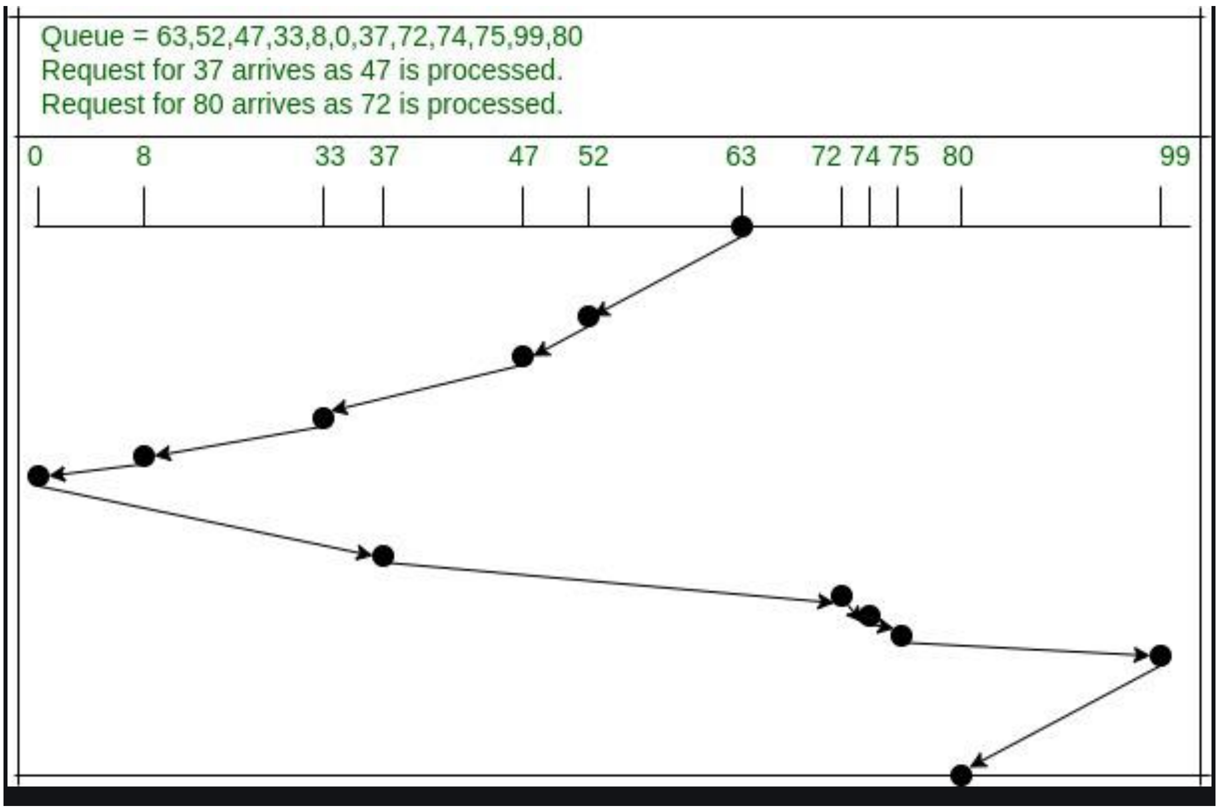
How it handles issue of "high variance in SSTF" ?

FSCAN addresses above mentioned issue by "freezing" queue once scan starts, requests that arrive after scan starts are processed in the next scan.

Performance analysis :

Citing theoretical analysis, it can be seen that SCAN results in lower average response time than FSCAN and higher average response time than shortest seek time first (SSTF). FSCAN algorithm has nice performance due to high throughput and low average response times. FSCAN removes problem of indefinite postponement.

Example : How requests are processed


Queue = 63,52,47,33,8,0,37,72,74,75,99,80
Request for 37 arrives as 47 is processed.
Request for 80 arrives as 72 is processed.
0
8
33
37
47
52
63
72
74
75
80
99