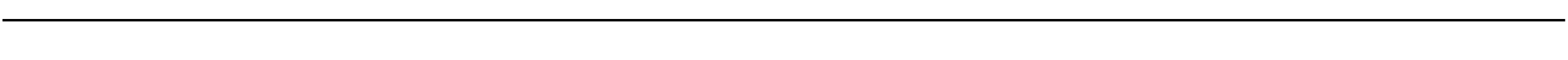
**Experiment 9**

**Title:** I/O Management: Disk scheduling

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

**Estimated time to complete this experiment:** 2 hours



**Objective:** Learning about Disk scheduling algorithms. Implementing a program for accessing various locations in the disk as per requirement.

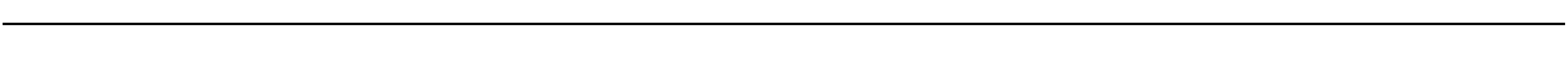


**Expected Outcome of Experiment:** To schedule I/O requests from various processes arriving for the disk to access data.



**Books/ Journals/ Websites referred:**

1. William Stallings, Operating System: Internals and Design Principles, Prentice Hall, 8thEdition, 2014, ISBN-10: 0133805913 • ISBN-13: 9780133805918.
2. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, John Wiley &Sons, Inc., 9thEdition, 2016, ISBN 978-81-265-5427-0



**Pre Lab/ Prior Concepts:** Any Programming platform.

**Brief description:**

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

* Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
* Two or more request may be far from each other so can result in greater disk arm movement.
* Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.



**New Concepts to be learned:** Disk scheduling algorithms.



**Requirements:** PC with any programming platform.



**Theory:**

**Disk Scheduling Algorithms**

* + - 1. **FCFS (First Come First Serve):**
* FCFS is the simplest of all the Disk Scheduling Algorithms.
* In FCFS, the requests are addressed in the order they arrive in the disk queue.

1. Advantages:
   1. Every request gets a fair chance
   2. No indefinite postponement
2. Disadvantages:
   1. Does not try to optimize seek time
   2. May not provide the best possible service
3. **SSTF (Shortest Seek Time First):**

* In SSTF, requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time.
* As a result, the request near the disk arm will get executed first.
* SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

1. Advantages:
   1. Average Response Time decreases
   2. Throughput increases
2. Disadvantages:
   1. Overhead to calculate seek time in advance
   2. Can cause Starvation for a request if it has higher seek time as compared to incoming requests
   3. High variance of response time as SSTF favours only some requests
3. **SCAN:**

* In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as elevator algorithm.
* As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

1. Advantages:
   1. High throughput
   2. Low variance of response time
   3. Average response time
2. Disadvantages:
   1. Long waiting time for requests for locations just visited by disk arm
3. **CSCAN (Circular SCAN):**

* In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.
* These situations are avoided in CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN.

1. Advantages:
   1. Provides more uniform wait time compared to SCAN



**Program:**

#include <iostream>

#include <algorithm>

#include <vector>

#include <cmath>

using namespace std;

void Take1D(int n, vector<int>& array)

{

    for (int i = 0; i < n; i++)

    {

        int temp;

        cin >> temp;

        array.push\_back(temp);

    }

}

void SCAN(int DiskSize, int NoOfrequest, vector<int> request)

{

    int current\_pos = 0;

    int distance = 0;

    cout << "Order of service: ";

    sort(request.begin(), request.end());

    auto it = lower\_bound(request.begin(), request.end(), current\_pos);

    int index = it - request.begin();

    for (int i = index; i < request.size(); i++)

    {

        distance += abs(request[i] - current\_pos);

        current\_pos = request[i];

        cout << current\_pos << " ";

    }

    for (int i = index - 1; i >= 0; i--)

    {

        distance += abs(request[i] - current\_pos);

        current\_pos = request[i];

        cout << current\_pos << " ";

    }

    cout << endl << "Total distance: " << distance << endl;

}

void CSCAN(int DiskSize, int NoOfrequest, vector<int> request)

{

    int current\_pos = 0;

    int distance = 0;

    cout << "Order of service: ";

    sort(request.begin(), request.end());

    auto it = lower\_bound(request.begin(), request.end(), current\_pos);

    int index = it - request.begin();

    for (int i = index; i < request.size(); i++)

    {

        distance += abs(request[i] - current\_pos);

        current\_pos = request[i];

        cout << current\_pos << " ";

    }

    distance += DiskSize - current\_pos;

    current\_pos = 0;

    cout << current\_pos << " ";

    for (int i = 0; i < index; i++)

    {

        distance += abs(request[i] - current\_pos);

        current\_pos = request[i];

        cout << current\_pos << " ";

    }

    cout << endl << "Total distance: " << distance << endl;

}

void SSTF(int DiskSize, int NoOfrequest, vector<int> request)

{

    int current\_pos = 0;

    int distance = 0;

    cout << "Order of service: ";

    while (!request.empty())

    {

        int closest\_pos = -1;

        int closest\_distance = INT32\_MAX;

        for (int i = 0; i < request.size(); i++)

        {

            int d = abs(request[i] - current\_pos);

            if (d < closest\_distance)

            {

                closest\_pos = i;

                closest\_distance = d;

            }

        }

        distance += closest\_distance;

        current\_pos = request[closest\_pos];

        cout << current\_pos << " ";

        request.erase(request.begin() + closest\_pos);

    }

    cout << endl << "Total distance: " << distance << endl;

}

void FCFSDS(int DiskSize, int NoOfrequest, vector<int> request)

{

    int current\_pos = 0;

    int distance = 0;

    cout << "Order of service: ";

    for (int i = 0; i < NoOfrequest; i++)

    {

        if (request[i] < 0 || request[i] > DiskSize - 1)

        {

            cout << "Error: Request " << request[i] << " is out of range!" << endl;

            continue;

        }

        distance += abs(request[i] - current\_pos);

        current\_pos = request[i];

        cout << current\_pos << " ";

    }

    cout << endl << "Total distance: " << distance << endl;

}

int main()

{

    vector<int> request;

    int DiskSize;

    int NoOfrequest;

    cout << "Enter the disk size: ";

    cin >> DiskSize;

    cout << "Enter the number of incoming requests: ";

    cin >> NoOfrequest;

    cout << "Enter the requests: ";

    Take1D(NoOfrequest, request);

    int choice;

    do

    {

        cout << endl << "Choose a disk scheduling algorithm: " << endl;

        cout << "1. First Come First Serve Disk Scheduling" << endl;

        cout << "2. Shortest Seek Time First Disk Scheduling" << endl;

        cout << "3. SCAN Disk Scheduling" << endl;

        cout << "4. C-SCAN Disk Scheduling" << endl;

        cout << "5. Exit" << endl;

        cout << "Enter your choice (1-5): ";

        cin >> choice;

        switch (choice)

        {

            case 1:

                FCFSDS(DiskSize, NoOfrequest, request);

                break;

            case 2:

                SSTF(DiskSize, NoOfrequest, request);

                break;

            case 3:

                SCAN(DiskSize, NoOfrequest, request);

                break;

            case 4:

                CSCAN(DiskSize, NoOfrequest, request);

                break;

            case 5:

                cout << "Exiting...";

                break;

            default:

                cout << "Invalid choice! Try again." << endl;

        }

    } while (choice != 5);

    return 0;

}

**Output:**

**Text

Description automatically generated**



**Conclusion:** Hence we have understoodorganization of files in multi-level directory structure.