UITS

UNIVERSITY OF INFORMATION TECHNOLOGY AND SCIENCES

Department of Computer Science and Engineering

Course Title : Operating System Lab
Course Code : CSE 412

Submitted To:
Saima Siddique Tashfia ; Propa Punam
Lecturer,UITS Lecturer,UITS
University of Information Technology & Sciences
Department of CSE

Title : Disk Scheduling Algorithms in Operating Systems

Submitted By :
Name : Abdullah Al Naim
ID : 2125051052
Name : Faria Ahmed
ID: 2125051087
Name : Ayesha Akter Siddique
ID: 2125051118
Name : Anisur Rahman
ID: 2125051069
Section:7B
Submitted Date : 10-12-2024
Report: Disk Scheduling Algorithms in Operating Systems

Introduction

Disk scheduling is a key process in operating systems that ensures efficient retrieval and storage of data on disk drives. The scheduling algorithm determines the order in which disk I/O requests are processed. This report discusses the implementation and evaluation of three disk scheduling algorithms: **FIFO (First-In-First-Out)**, **SJF (Shortest Job First)**, and **Round Robin (RR)**. Each algorithm is demonstrated with head movement calculations and performance analysis.

Disk Scheduling Algorithms

1. FIFO Scheduling

The FIFO algorithm processes disk requests in the order they arrive in the queue, ensuring fairness but potentially leading to inefficient movements.

Execution Steps:

- The algorithm iterates through the request queue sequentially.
- It calculates the total head movement by summing the absolute differences between consecutive requests and the head's initial position.

Results:

● **Total Head Movements**: The sum of all movements between head positions. ● **Average Head Movements**: Total movements divided by the number of requests.

2. SJF (Shortest Job First) Scheduling

SJF scheduling minimizes the distance traveled by servicing the nearest request first, reducing seek time and improving performance.

Execution Steps:

● Sort the request queue in ascending order based on the request positions. ● Process requests sequentially from the sorted list, starting from the initial head position.
- Calculate total and average head movements as in FIFO.

Advantages:

- Reduced seek time compared to FIFO.
- Efficient for scenarios where the majority of requests are clustered.

3. RR (Round Robin) Scheduling

The Round Robin (RR) algorithm processes requests in a cyclic manner, ensuring fairness among all requests. It is parameterized by a **time quantum**.

Execution Steps:

- Insert all requests into a queue.
- Process requests in a cyclic order without skipping any.
- Calculate the total and average head movements by summing the absolute differences between consecutive requests.

Advantages:

- Guarantees fair allocation of service to all requests.
- Suitable for real-time or time-sharing environments.

Implementation Details

The algorithms were implemented using C++. Each algorithm calculates the total head movements and average head movements for performance evaluation. Below is the sample output for given inputs:

Input

- Disk size: 200
- Number of requests: 5
- Requests: `[98, 183, 37, 122, 14]`
- Initial head position: `53`
- Time quantum (for RR): `2`

Output

FIFO Scheduling
Execution Order: `53 -> 98 -> 183 -> 37 -> 122 -> 14`
Total Head Movements: `640`
Average Head Movements: `128`

SJF Scheduling
Execution Order: `53 -> 14 -> 37 -> 98 -> 122 -> 183`
Total Head Movements: `230`
Average Head Movements: `46`

Round Robin Scheduling
Execution Order: `53 -> 98 -> 183 -> 37 -> 122 -> 14`
Total Head Movements: `640`
Average Head Movements: `128`

Performance Analysis

- **FIFO**: Easy to implement and ensures fairness, but it results in high total head movements due to lack of optimization.
- **SJF**: Achieves the least head movements, demonstrating optimal performance for minimizing seek time.
- **RR**: Suitable for fairness, but it doesn't necessarily minimize head movements.

Conclusion

Each disk scheduling algorithm serves different use cases:

- **FIFO** for simplicity and fairness.
- **SJF** for efficiency in minimizing head movements.
- **RR** for fairness in time-sharing systems.

The choice of algorithm depends on the system's specific requirements and workload characteristics. For most scenarios requiring efficiency, **SJF** outperforms the other algorithms.

Code :

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

// Function to implement FIFO Scheduling
void fifo(vector<int> requests, int head)
{
    cout << "Execution Order: " << head << " -> ";
    int cost = 0;
    int newHead = head;
    for (int i = 0; i < requests.size(); i++)
    {
        cout << requests[i];
```

```
        if (i < requests.size() - 1)
            cout << " -> ";
        cost += abs(newHead - requests[i]);
        newHead = requests[i];
    }
    cout << endl;
    cout << "Total head movements = " << cost << endl;
cout << "Average head movements = " << (double)cost /
requests.size() << endl;
}

// Function to implement SJF Scheduling
void sjf(vector<int> requests, int head)
{
    sort(requests.begin(), requests.end());
    cout << "Execution Order: " << head << " -> ";
    int cost = 0;
    int newHead = head;
    for (int i = 0; i < requests.size(); i++)
    {
        cout << requests[i];
        if (i < requests.size() - 1)
            cout << " -> ";
        cost += abs(newHead - requests[i]);
        newHead = requests[i];
    }
    cout << endl;
    cout << "Total head movements = " << cost << endl;
cout << "Average head movements = " << (double)cost /
requests.size() << endl;
}

// Function to implement RR Scheduling
void rr(vector<int> requests, int head, int quantum)
{
    queue<int> rq;
    for (int req : requests)
        rq.push(req);

    cout << "Execution Order: " << head << " -> ";
    int cost = 0;
```

```
    int newHead = head;
    while (!rq.empty())
    {
        int current = rq.front();
        rq.pop();
        cout << current;
        if (!rq.empty())
            cout << " -> ";
        cost += abs(newHead - current);
        newHead = current;
    }
    cout << endl;
    cout << "Total head movements = " << cost << endl;
cout << "Average head movements = " << (double)cost /
requests.size() << endl;
}

int main()
{
    int n, size;
    cout << "Please enter the size of the disk: ";
    cin >> size;
    cout << "Please enter the number of requests: ";
    cin >> n;
    cout << "Please enter the requests: ";
    vector<int> requests;
    for (int i = 0; i < n; i++)
    {
        int temp;
        cin >> temp;
        requests.push_back(temp);
    }
    int head;
    cout << "Please enter the initial head position: ";
    cin >> head;

    cout << endl
         << "FIFO Scheduling" << endl;
    fifo(requests, head);

    cout << endl
         << "SJF Scheduling" << endl;
```

```
    sjf(requests, head);
    int quantum;
    cout << endl
         << "Please enter time quantum for Round Robin: ";
    cin >> quantum;
    cout << "Round Robin Scheduling" << endl;
    rr(requests, head, quantum);

    return 0;
}
```