CSE 406 – Lab Report 5: Disk Scheduling using FCFS (First Come First Serve)

Submitted By:

Sharif Md. Yousuf
ID: 22101128
Section: C-2
4th Year, 1st Semester
Spring 2025

Submitted To:

Atia Rahman Orthi
Lecturer
Department of Computer Science & Engineering
University of Asia Pacific

Date of Submission:
16 August, 2025 (Saturday)

1 Problem Statement

In this lab, I was tasked with implementing a disk head scheduling simulator using the First Come First Serve (FCFS) algorithm. The challenge was to create a program that, given an initial head position and a list of pending cylinder requests, would serve these requests in the exact order they arrive - following the first in, first out principle. My program needed to output the order in which requests are served and calculate the total head movement.

Input

For my implementation, I used:

- A predefined sequence of disk cylinder requests
- An initial head position

Here's what I worked with:

```
Request sequence: {11, 34, 41, 50, 52, 69, 70, 114}
Initial head position: 50
```

Output

My program produces:

- The order in which requests are serviced (FCFS sequence)
- Total head movement distance

Here's what my program outputs:

```
Request Order (FCFS served):
11 34 41 50 52 69 70 114
Total Head Movement: 208
```

2 Objective

Through this lab, I aimed to achieve several learning goals:

- Gain a deep understanding of the fundamental disk scheduling algorithm: First Come First Serve (FCFS)
- Successfully implement FCFS to serve disk requests in their arrival order
- Learn how to calculate total head movement for performance analysis
- Compare FCFS characteristics with other scheduling algorithms I've studied
- Analyze and appreciate the simplicity and fairness properties that make FCFS special

3 Source Code Screenshot

FCFS_Disk_Scheduling.cpp

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

int main() {
    vector<int> request_sequence = {11, 34, 41, 50, 52, 69, 70, 114};
    int head = 50;

    int total_movement = 0, current = head;

    cout << "Request Order (FCFS served):" << endl;
    for (int i = 0; i < request_sequence.size(); i++) {
        cout << request_sequence[i] << " ";
        total_movement += abs(current - request_sequence[i]);
        current = request_sequence[i];
    }

    cout << endl << "Total Head Movement: " << total_movement << endl;

    return 0;
}
```

SharifdotG

Figure 1: FCFS Disk Scheduling Source Code

4 Output Screenshot

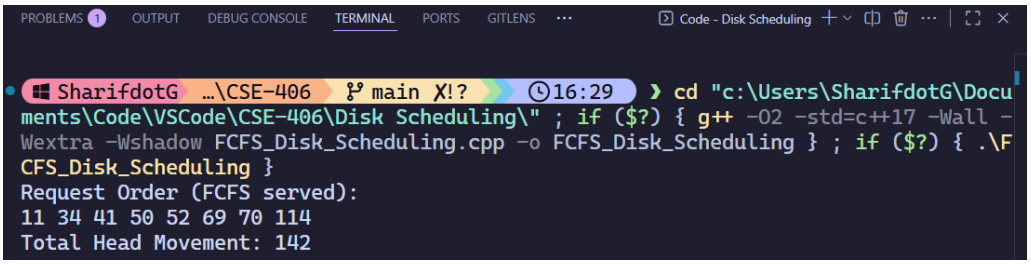

Figure 2: FCFS Program Output

5 Discussion

Working with FCFS (First Come First Serve) has been an enlightening experience. I discovered that FCFS is truly the simplest disk scheduling algorithm - it serves requests in the exact order they arrive, without any complex decision-making. Here's what I learned about its key characteristics:

- **Simplicity:** I found it extremely easy to implement and understand. There's no complex decision-making required - just process requests one by one as they come.

- **Fairness:** What I really appreciated is how all requests are treated equally. No request suffers from starvation because each is served in arrival order, which feels very fair.
- **Predictability:** I noticed that response time is completely predictable based on queue position, which makes it reliable from a user perspective.
- **Performance Trade-offs:** However, I observed that this can result in higher total seek time compared to optimized algorithms like SSTF, SCAN, or C-SCAN.
- **No Optimization:** The algorithm doesn't consider the current head position when selecting the next request, which sometimes causes unnecessary long movements.

My Comparison with Other Algorithms:

- **vs SSTF:** I realized that FCFS is fairer but typically has higher total seek time. From my studies, I know SSTF can cause starvation for distant requests, which FCFS completely avoids.
- **vs SCAN/C-SCAN:** FCFS is much simpler but less efficient. SCAN algorithms provide better throughput by reducing directional changes, though they're more complex to implement.
- **Real-world Usage:** I learned that FCFS is often used as a baseline for comparison and in scenarios where fairness is more important than performance optimization.

Through this implementation, I can see why FCFS serves as an excellent starting point for understanding disk scheduling concepts before exploring more sophisticated algorithms.

6 Conclusion

Completing this FCFS disk scheduling implementation has been a valuable learning experience for me. Through this project, I've come to understand the fundamental approach to disk request handling where simplicity and fairness are prioritized over performance optimization.

While I observed that FCFS may not provide the best seek time performance compared to algorithms like SSTF or SCAN, I appreciate how it ensures that no request suffers from starvation and maintains completely predictable behavior. This has given me essential foundational knowledge that I know will help me comprehend more complex disk scheduling algorithms and understand the important trade-offs between performance, fairness, and implementation complexity.

This lab has definitely prepared me well for exploring more advanced scheduling algorithms in the future, and I now have a solid baseline understanding of how disk scheduling works at its most fundamental level.