LMS Assignment Report

Department: Department of Computer Science & Engineering
Course: ICS1313 - Operating System Practices Laboratory
Assignment No: 10
Name: Simiyon Vinscent Samuel L
Reg No: 3122247001062

Assignment Title

Advanced Disk Scheduling Algorithms: SSTF and C-SCAN Implementation

Aim / Learning Objectives

To implement and perform comprehensive analysis of the Shortest Seek Time First (SSTF) and Circular SCAN (C-SCAN) disk scheduling algorithms. Primary objectives include:

- Mastering fundamental disk scheduling concepts and understanding the critical importance of seek time optimization in storage systems
- Implementing SSTF algorithm that intelligently selects the closest request to minimize total seek time
- Developing C-SCAN algorithm which moves the disk head in one direction while servicing requests in a circular manner
- Conducting thorough analysis of total seek time as the key performance metric for disk scheduling efficiency
- Applying advanced sorting and selection techniques to solve complex scheduling optimization problems

Theory

Disk scheduling algorithms are essential components of operating systems that manage how disk I/O requests are serviced to optimize system performance. The primary goal is to minimize seek time - the time required for the disk head to move from its current position to the desired track.

SSTF (Shortest Seek Time First): This algorithm selects the request closest to the current head position, minimizing individual seek distances. While it reduces average seek time, it may cause starvation for requests at disk extremes.

C-SCAN (Circular SCAN): This algorithm moves the head in one direction, servicing all requests until reaching the disk end, then jumps to the beginning and continues. This provides more uniform wait times compared to traditional SCAN algorithms.

Algorithm

SSTF Algorithm:

1. Initialize current head position and total seek time counter
2. While unprocessed requests remain:
    - Find the request with minimum distance from current head position
    - Add the seek distance to total seek time
    - Move head to the selected request position
    - Mark request as completed
3. Return total seek time

C-SCAN Algorithm:

1. Sort all requests in ascending order
2. Find the first request greater than or equal to current head position
3. Service all requests from current position to disk end
4. If requests remain, jump to disk beginning (track 0)
5. Service remaining requests from beginning
6. Calculate and return total seek time

Implementation

```
#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;

int findClosestRequest(int requests[], int n, int head) {
    int minDistance = 1000000, selectedIndex = -1;

    for (int i = 0; i &lt; n; i++) {
        if (requests[i] != -1) {
            int distance = abs(requests[i] - head);
            if (distance &lt; minDistance) {
                minDistance = distance;
                selectedIndex = i;
            }
        }
    }
    return selectedIndex;
}

void implementSSTF(int requests[], int n, int head) {
    int totalSeekTime = 0, currentHead = head, servicedRequests = 0;

    printf("SSTF Disk Scheduling:\n");
    printf("Head Movement Sequence: %d", head);
```

```
    while (servicedRequests &lt; n) {
        int selectedIndex = findClosestRequest(requests, n, currentHead);
        if (selectedIndex == -1) break;

        totalSeekTime += abs(requests[selectedIndex] - currentHead);
        currentHead = requests[selectedIndex];
        printf(" -&gt; %d", currentHead);

        requests[selectedIndex] = -1; // Mark as processed
        servicedRequests++;
    }

    printf("\nTotal Seek Time = %d\n", totalSeekTime);
}

int compareFunction(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

void implementCSCAN(int requests[], int n, int head, int diskSize) {
    int totalSeekTime = 0, currentHead = head;

    qsort(requests, n, sizeof(int), compareFunction);

    printf("C-SCAN Disk Scheduling:\n");
    printf("Head Movement Sequence: %d", head);

    // Find first request &gt;= head position
    int splitIndex;
    for (splitIndex = 0; splitIndex &lt; n; splitIndex++) {
        if (requests[splitIndex] &gt;= head) break;
    }

    // Service requests from head to end
    for (int j = splitIndex; j &lt; n; j++) {
        totalSeekTime += abs(requests[j] - currentHead);
        currentHead = requests[j];
        printf(" -&gt; %d", currentHead);
    }

    // Jump to beginning if requests remain
    if (splitIndex &gt; 0) {
        totalSeekTime += abs(diskSize - 1 - currentHead); // Move to end
        totalSeekTime += diskSize - 1; // Jump to beginning
        currentHead = 0;
        printf(" -&gt; %d (jump)", currentHead);

        // Service remaining requests from beginning
        for (int j = 0; j &lt; splitIndex; j++) {
            totalSeekTime += abs(requests[j] - currentHead);
            currentHead = requests[j];
            printf(" -&gt; %d", currentHead);
        }
    }

    printf("\nTotal Seek Time = %d\n", totalSeekTime);
```

```
}

int main() {
    int n, head, diskSize, choice;

    printf("Enter number of disk requests: ");
    scanf("%d", &amp;n);

    int requests[n], requestsCopy[n];

    printf("Enter the disk requests: ");
    for (int i = 0; i &lt; n; i++) {
        scanf("%d", &amp;requests[i]);
        requestsCopy[i] = requests[i];
    }

    printf("Enter initial head position: ");
    scanf("%d", &amp;head);

    printf("Enter total disk size: ");
    scanf("%d", &amp;diskSize);

    do {
        printf("\n********** DISK SCHEDULING MENU **********\n");
        printf("1. SSTF Algorithm\n");
        printf("2. C-SCAN Algorithm\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &amp;choice);

        switch (choice) {
            case 1:
                printf("\n=== SSTF Algorithm Execution ===\n");
                implementSSTF(requestsCopy, n, head);
                // Restore original requests for next algorithm
                for (int i = 0; i &lt; n; i++) {
                    requestsCopy[i] = requests[i];
                }
                break;

            case 2:
                printf("\n=== C-SCAN Algorithm Execution ===\n");
                implementCSCAN(requestsCopy, n, head, diskSize);
                // Restore original requests
                for (int i = 0; i &lt; n; i++) {
                    requestsCopy[i] = requests[i];
                }
                break;

            case 3:
                printf("Program terminated successfully.\n");
                break;

            default:
                printf("Invalid choice! Please select 1, 2, or 3.\n");
        }
```

```
    } while (choice != 3);

    return 0;
}
```

Test Cases

Test Scenario	Input Configuration	Expected Results
Standard Test	Requests: 98,183,37,122,14,124,65,67 Head: 53 Disk Size: 200	SSTF Total Seek Time = 236 C-SCAN Total Seek Time = 322
Edge Case	Requests: 10,199,50,150 Head: 100 Disk Size: 200	SSTF optimizes local movements C-SCAN provides uniform service
Performance Analysis	Various request patterns	SSTF: Lower average seek time C-SCAN: Better worst-case performance

Performance Analysis

SSTF Algorithm:

- **Advantages:** Minimizes total seek time, optimal for random access patterns
- **Disadvantages:** May cause starvation for requests at disk extremes
- **Time Complexity:** O(n^2) for request selection process
- **Best Use Case:** Systems with moderate request loads and no strict fairness requirements

C-SCAN Algorithm:

- **Advantages:** Provides uniform wait times, eliminates starvation
- **Disadvantages:** Higher total seek time compared to SSTF
- **Time Complexity:** O(n log n) due to sorting requirement
- **Best Use Case:** Systems requiring fair access and predictable response times

Learning Outcomes

- Successfully implemented and analyzed two fundamental disk scheduling algorithms
- Mastered the trade-offs between seek time optimization and fairness in resource allocation
- Developed proficiency in system-level programming using C language
- Gained practical experience with sorting algorithms and their application in system optimization
- Understood the performance implications of different scheduling strategies in storage systems

- Learned to measure and compare algorithm efficiency using quantitative metrics

Conclusion

This assignment provided comprehensive hands-on experience with disk scheduling algorithms crucial for operating system performance. The implementation demonstrated that SSTF excels in minimizing total seek time but may sacrifice fairness, while C-SCAN ensures uniform service distribution at the cost of increased seek time. The choice between algorithms depends on specific system requirements and performance priorities.

References

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
2. Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.