...for a complete human being

Texas International College

Lab No: 10 **Date: 2082/**

Title: Write a program to calculate the seek time for user input pending request, total no of cylinders and current position of I/O read/write head using FCFS disk scheduling algorithm.

The First Come First Serve (FCFS) disk scheduling algorithm is the simplest and most straightforward method. In FCFS, disk requests are processed strictly in the order they arrive in the queue, similar to a normal service line. This ensures fairness because no request is skipped or postponed.

Algorithm:

Step 1: Input the total number of cylinders in the disk.

Step 2: Input the current position of the disk head.

Step 3: Input the request queue (list of pending requests).

Step 4: Initialize total_seek = 0.

Step 5: For each request in the order it appears:

- Calculate seek = |current_position – next_request|.
- Add this seek to total_seek.
- Move the head to the new position.

Step 6: Repeat until all requests are serviced.

Step 7: Output the total seek time and the order of servicing.

Language: C++

IDE: VS Code

Code:

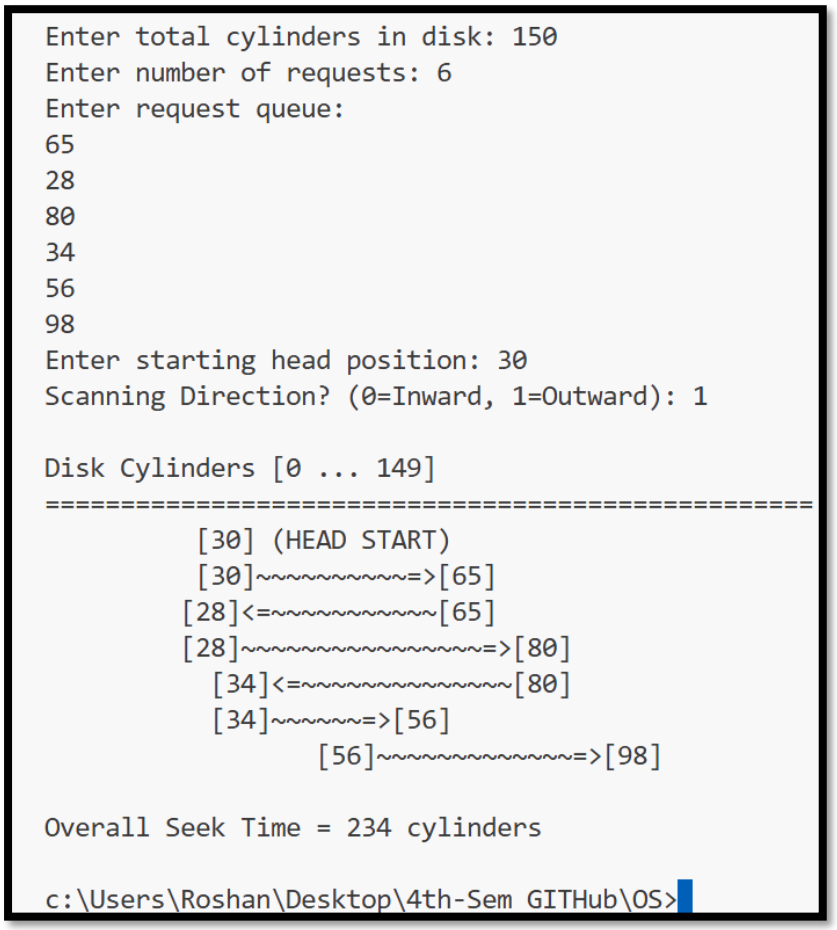
```
#include <stdio.h>
#include <stdlib.h>

#define SCALE_LENGTH 50  // changed scale length
void printScale(int max_cylinders) {
    printf("\nDisk Cylinders [0 ... %d]\n", max_cylinders);
    for (int i = 0; i <= SCALE_LENGTH; i++) printf("=");
    printf("\n");
}
int getScaledPosition(int pos, int max_cylinders) {
    return (pos * SCALE_LENGTH) / max_cylinders;
}
void printStep(int from, int to, int max_cylinders) {
    int from_pos = getScaledPosition(from, max_cylinders);
    int to_pos   = getScaledPosition(to, max_cylinders);
    if (to_pos >= from_pos) {
        // Forward move
        for (int i = 0; i < from_pos; i++) printf(" ");
        printf("[%d]", from);
        for (int i = from_pos + 1; i < to_pos; i++) printf("~");
        printf("=>[%d]\n", to);
    } else {
        // Backward move
        for (int i = 0; i < to_pos; i++) printf(" ");
```

```
            printf("[%d]", to);
            printf("<=");
            for (int i = to_pos + 1; i < from_pos; i++) printf("~");
            printf("[%d]\n", from);
        }
    }
    int main() {
        int cylinders, n, head;
        int *requests;
        printf("Enter total cylinders in disk: ");
        scanf("%d", &cylinders);
        printf("Enter number of requests: ");
        scanf("%d", &n);
        requests = (int *)malloc(n * sizeof(int));
        printf("Enter request queue:\n");
        for (int i = 0; i < n; i++) {
            scanf("%d", &requests[i]);
            if (requests[i] < 0 || requests[i] >= cylinders) {
                printf("Invalid request %d! (valid range: 0 - %d)\n", requests[i], cylinders - 1);
                free(requests);
                return 1;
            }
        }
        printf("Enter starting head position: ");
        scanf("%d", &head);
```

```
    if (head < 0 || head >= cylinders) {
        printf("Invalid starting position.\n");
        free(requests);
        return 1;
    }
    // Direction input (not used in FCFS but kept for similarity)
    int dir;
    printf("Scanning Direction? (0=Inward, 1=Outward): ");
    scanf("%d", &dir);
    printScale(cylinders - 1);
    int total_seek = 0;
    int current = head;
    int init_pos = getScaledPosition(current, cylinders);
    for (int i = 0; i < init_pos; i++) printf(" ");
    printf("[%d] (HEAD START)\n", current);
    for (int i = 0; i < n; i++) {
        printStep(current, requests[i], cylinders);
        total_seek += abs(requests[i] - current);
        current = requests[i];
    }
    printf("\nOverall Seek Time = %d cylinders\n", total_seek);
    free(requests);
    return 0;
}
```

Output:



Conclusion:

The FCFS disk scheduling algorithm is simple and fair since requests are processed in the order they arrive. However, it does not minimize seek time and may cause unnecessary head movement if requests are far apart. Despite this drawback, FCFS is widely used for its simplicity and fairness in servicing disk I/O requests.