Lab No: 11 **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 SCAN disk scheduling algorithm.

In SCAN, the disk arm moves in one direction (towards higher or lower cylinder numbers), servicing all pending requests until it reaches the end of the disk. Then, it reverses direction and services the remaining requests on its way back. This back-and-forth movement is similar to an elevator, which explains the name. The advantage of SCAN over FCFS is that it reduces variance in response time and ensures that all requests are serviced in an orderly manner.

Algorithm:

Step 1: Input total cylinders, request queue, head position, and direction.

Step 2: Sort the request queue in ascending order.

Step 3: Find the position of the head in the sorted queue.

Step 4:

- If moving right: service requests to the right, go to last cylinder, then reverse and service left.
- If moving left: service requests to the left, go to first cylinder, then reverse and service right.

Step 5: For each movement, calculate seek distance and update total seek time.

Step 6: Output the servicing order and total seek distance.

Language: C++

IDE: VS Code

Code:

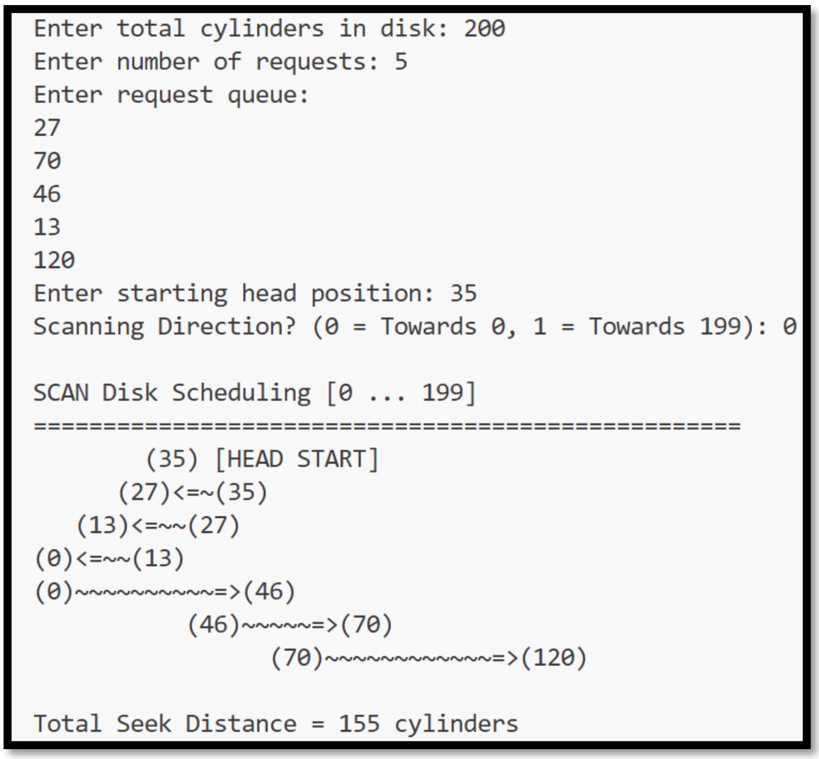
```
#include <stdio.h>

#include <stdlib.h>


#define SCALE_LENGTH 50  // keep same scale length

void printScale(int max_cylinders) {

    printf("\nSCAN Disk Scheduling [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;
int dir;
printf("Scanning Direction? (0 = Towards 0, 1 = Towards %d): ", cylinders - 1);
scanf("%d", &dir);
// Sort requests
for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
        if (requests[i] > requests[j]) {
            int temp = requests[i];
            requests[i] = requests[j];
            requests[j] = temp;
        }
    }
}
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);
// find the split point
int index = 0;
```

```
while (index < n && requests[index] < head) index++;

if (dir == 1) {

    // Move outward first

    for (int i = index; i < n; i++) {

        printStep(current, requests[i], cylinders);

        total_seek += abs(requests[i] - current);

        current = requests[i];

    }

    // then to the end

    if (current != cylinders - 1) {

        printStep(current, cylinders - 1, cylinders);

        total_seek += abs((cylinders - 1) - current);

        current = cylinders - 1;

    }

    // now reverse and go left

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

        printStep(current, requests[i], cylinders);

        total_seek += abs(requests[i] - current);

        current = requests[i];

    }

} else {

    // Move inward first

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

        printStep(current, requests[i], cylinders);

        total_seek += abs(requests[i] - current);
```

```
            current = requests[i];
        }
        // then to the beginning
        if (current != 0) {
            printStep(current, 0, cylinders);
            total_seek += current; // distance to 0
            current = 0;
        }
        // now reverse and go right
        for (int i = index; i < n; i++) {
            printStep(current, requests[i], cylinders);
            total_seek += abs(requests[i] - current);
            current = requests[i];
        }
    }
    printf("\nTotal Seek Distance = %d cylinders\n", total_seek);
    free(requests);
    return 0;
}
```

Output:



```
Enter total cylinders in disk: 200
Enter number of requests: 5
Enter request queue:
27
70
46
13
120
Enter starting head position: 35
Scanning Direction? (0 = Towards 0, 1 = Towards 199): 0

SCAN Disk Scheduling [0 ... 199]
===================================================
        (35) [HEAD START]
      (27)<=~(35)
   (13)<=~~(27)
(0)<=~~(13)
(0)~~~~~~~~~~=>(46)
           (46)~~~~~=>(70)
                 (70)~~~~~~~~~~~~=>(120)

Total Seek Distance = 155 cylinders
```

Conclusion:

The SCAN disk scheduling algorithm provides a fair balance between efficiency and simplicity. Unlike FCFS, it reduces the total seek time by servicing requests in a structured path, similar to an elevator. Although requests at the edges of the disk may experience longer delays, SCAN ensures that no request is starved and that disk head movement is more optimized than FCFS.