...for a complete human being

Texas International College

Lab No: 13 **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 LOOK disk scheduling algorithm.

Disk scheduling algorithms manage the order in which pending I/O requests are serviced to improve performance. The LOOK algorithm is a variation of the SCAN algorithm. Unlike SCAN, which moves the head to the end of the disk regardless of requests, LOOK only goes as far as the last request in each direction, then reverses. This reduces unnecessary head movement and improves seek time. LOOK is efficient for systems with many requests spread across disk cylinders.

Algorithm:

Step 1: Input total number of cylinders, current head position, and pending requests.

Step 2: Divide the requests into two groups: requests less than and greater than the current head position.

Step 3: Sort both groups in ascending order.

Step 4: Move the head in one direction (e.g., towards higher cylinders) servicing requests until the last request in that direction.

Step 5: Reverse direction and service the remaining requests.

Step 6: Calculate total seek time as the sum of absolute distances moved by the head.

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

Language: C++

IDE: VS Code

Code:

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

#define SCALE_LENGTH 50  // scale length for visualization
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) {
        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 {
        for (int i = 0; i < to_pos; i++) printf(" ");
        printf("[%d]", to);
        printf("<=");
```

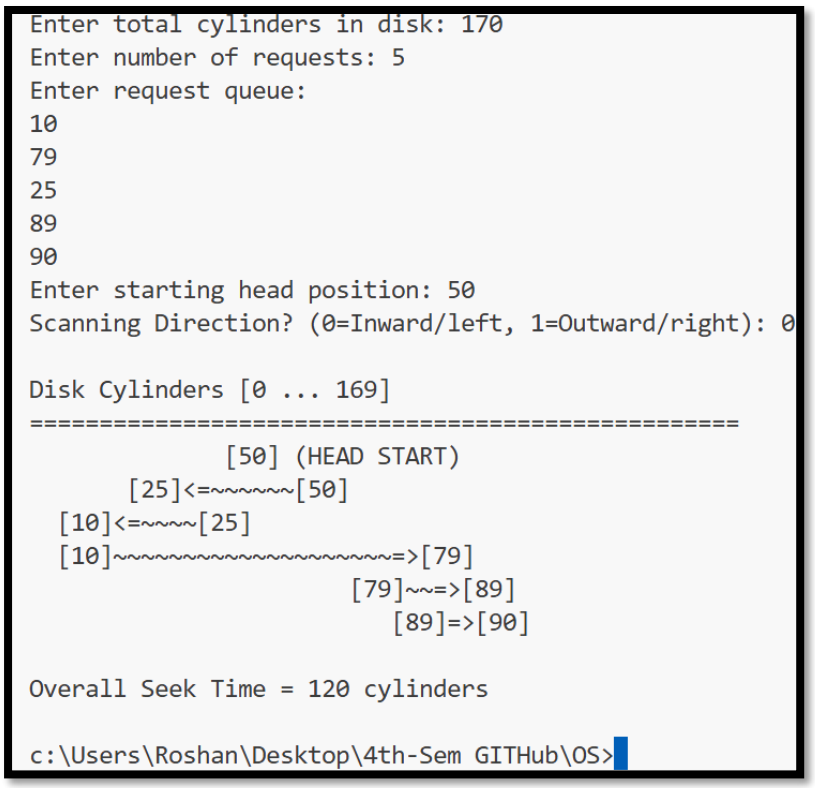
```
        for (int i = to_pos + 1; i < from_pos; i++) printf("~");
        printf("[%d]\n", from);
    }
}
int cmpfunc(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}
int main() {
    int cylinders, n, head;
    int *requests;
    printf("Enter total cylinders in disk: ");
    scanf("%d", &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=Inward/left, 1=Outward/right): ");
scanf("%d", &dir);
// Sort requests for LOOK order
qsort(requests, n, sizeof(int), cmpfunc);
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);
// LOOK Scheduling
if (dir == 1) {
    // Outward/right first
    for (int i = 0; i < n; i++) {
        if (requests[i] >= head) {
            printStep(current, requests[i], cylinders);
            total_seek += abs(requests[i] - current);
            current = requests[i];
```

```
            }
        }
        // then reverse direction (without going to the end of disk)
        for (int i = n - 1; i >= 0; i--) {
            if (requests[i] < head) {
                printStep(current, requests[i], cylinders);
                total_seek += abs(requests[i] - current);
                current = requests[i];
            }
        }
    } else {
        // Inward/left first
        for (int i = n - 1; i >= 0; i--) {
            if (requests[i] <= head) {
                printStep(current, requests[i], cylinders);
                total_seek += abs(requests[i] - current);
                current = requests[i];
            }
        }
        // then reverse direction
        for (int i = 0; i < n; i++) {
            if (requests[i] > head) {
                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 LOOK algorithm efficiently reduces seek time compared to FCFS and SCAN by avoiding unnecessary movement to disk ends. It provides a balanced performance for systems with multiple pending I/O requests, minimizing average head movement while servicing requests in an orderly manner.