

| **TITLE: Disk Scheduling Algorithms** |
| --- |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Disk Scheduling Algorithm like FCFS, SSTF, SCAN, CSCAN, LOOK.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 4.** To understand various Memory, I/O and File management techniques.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

* Knowledge of disk scheduling algorithm.
* Calculation of seek time and transfer time etc.

**Description of the application to be implemented**:

### 1. First-Come, First-Served (FCFS)

* **Description**: The simplest scheduling algorithm, FCFS processes requests in the order they arrive.
* **Implementation**:
  + Maintain a queue of incoming disk requests.
  + When the disk head is idle, it serves the request at the front of the queue.
* **Advantages**: Easy to implement and understand.
* **Disadvantages**: Can lead to the "convoy effect," where short requests get stuck behind long ones, increasing average wait time.

### 2. Shortest Seek Time First (SSTF)

* **Description**: SSTF selects the request that is closest to the current head position, reducing seek time.
* **Implementation**:
  + Maintain a list of pending requests and their distances from the current head position.
  + Always serve the closest request next.
* **Advantages**: Reduces average seek time compared to FCFS.
* **Disadvantages**: Can lead to starvation for requests far from the head’s current position.

### 3. SCAN (Elevator Algorithm)

* **Description**: The SCAN algorithm moves the disk arm in one direction, servicing requests until it reaches the end, then reverses direction.
* **Implementation**:
  + Maintain a sorted list of requests.
  + Start scanning in one direction, serving all requests until the end is reached, then reverse direction.
* **Advantages**: More equitable than SSTF; reduces waiting time for requests located at the ends.
* **Disadvantages**: Average wait time can still be high for requests located just outside the current path.

### 4. Circular SCAN (CSCAN)

* **Description**: Similar to SCAN, but once the arm reaches the end, it quickly returns to the beginning without servicing any requests on the return trip.
* **Implementation**:
  + Sort requests and move in one direction, servicing them until the end.
  + Jump back to the start of the disk and continue servicing in the same direction.
* **Advantages**: Provides a more uniform wait time, as requests at the ends are treated similarly to those in the middle.
* **Disadvantages**: The quick return can lead to increased average wait times for requests at the end of the disk.

### 5. LOOK

* **Description**: LOOK is a variation of SCAN, where the disk arm only goes as far as the last request in either direction before reversing.
* **Implementation**:
  + Maintain a sorted list of requests and scan only up to the last request in the current direction.
* **Advantages**: Reduces unnecessary travel time and can improve average wait time compared to SCAN.
* **Disadvantages**: Similar to SCAN, can still have long waits for requests located just outside the path of the head movement.

**Implementation details:**

Elevator (SCAN):

#include <stdio.h>

#include <stdlib.h>

void SCAN(int requests[], int size, int head, int direction) {

int seek\_sequence[size + 2]; // +2 for head start and end

int index = 0;

// Sort requests

for (int i = 0; i < size; i++) {

for (int j = i + 1; j < size; j++) {

if (requests[i] > requests[j]) {

int temp = requests[i];

requests[i] = requests[j];

requests[j] = temp;

}

}

}

// Add the head to the seek sequence

seek\_sequence[index++] = head;

// Move towards the end

if (direction == 1) {

for (int i = 0; i < size; i++) {

if (requests[i] >= head) {

seek\_sequence[index++] = requests[i];

}

}

seek\_sequence[index++] = 100; // Assuming 100 is the end of the disk

// Now move towards the beginning

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

if (requests[i] < head) {

seek\_sequence[index++] = requests[i];

}

}

} else {

// Move towards the beginning first

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

if (requests[i] <= head) {

seek\_sequence[index++] = requests[i];

}

}

seek\_sequence[index++] = 0; // Assuming 0 is the start of the disk

// Now move towards the end

for (int i = 0; i < size; i++) {

if (requests[i] > head) {

seek\_sequence[index++] = requests[i];

}

}

}

// Print the seek sequence

printf("SCAN Seek Sequence: ");

for (int i = 0; i < index; i++) {

printf("%d ", seek\_sequence[i]);

}

printf("\n");

}

int main() {

int requests[] = { 34, 78, 12, 50, 24, 92 };

int size = sizeof(requests) / sizeof(requests[0]);

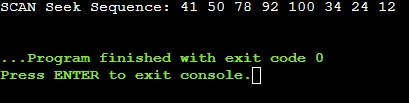
int head = 41; // Initial head position

int direction = 1; // 1 for up, 0 for down

SCAN(requests, size, head, direction);

return 0;

}



CSAN:

#include <stdio.h>

#include <stdlib.h>

void C\_SCAN(int requests[], int size, int head, int direction) {

int seek\_sequence[size + 2]; // +2 for head start and end

int index = 0;

// Sort requests

for (int i = 0; i < size; i++) {

for (int j = i + 1; j < size; j++) {

if (requests[i] > requests[j]) {

int temp = requests[i];

requests[i] = requests[j];

requests[j] = temp;

}

}

}

// Add the head to the seek sequence

seek\_sequence[index++] = head;

if (direction == 1) {

// Move towards the end

for (int i = 0; i < size; i++) {

if (requests[i] >= head) {

seek\_sequence[index++] = requests[i];

}

}

seek\_sequence[index++] = 100; // Jump to the end

// Jump to the beginning

for (int i = 0; i < size; i++) {

if (requests[i] < head) {

seek\_sequence[index++] = requests[i];

}

}

} else {

// Move towards the beginning first

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

if (requests[i] <= head) {

seek\_sequence[index++] = requests[i];

}

}

seek\_sequence[index++] = 0; // Jump to the start

// Jump to the end

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

if (requests[i] > head) {

seek\_sequence[index++] = requests[i];

}

}

}

// Print the seek sequence

printf("C-SCAN Seek Sequence: ");

for (int i = 0; i < index; i++) {

printf("%d ", seek\_sequence[i]);

}

printf("\n");

}

int main() {

int requests[] = { 34, 78, 12, 50, 24, 92 };

int size = sizeof(requests) / sizeof(requests[0]);

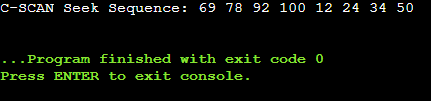
int head = 69; // Initial head position

int direction = 1; // 1 for up, 0 for down

C\_SCAN(requests, size, head, direction);

return 0;

}



**Conclusion**:

**Post Lab Descriptive Questions**

1. A disk drive has 200 cylinders numbered from 0 to 199. The disk head is initially at cylinder 53. The queue of pending requests in FIFO order is :

98, 183, 37, 122, 14, 124, 65, 67.

Starting from the current head position, what is the total distance travelled (in cylinders) by disk arm to satisfy the requests using CSCAN and Look. Illustrate with figures in each case.

**Post Lab Objective Questions**

1. In a hard disk, what rotates about a central spindle
   1. Disk
   2. Platter
   3. Sector
   4. None of the above

**Ans: b. Platter**

1. The time required to move the disk arm to the required track is known as
   1. Latency time
   2. Access time
   3. Seek time
   4. None of the above

**Ans: c. Seek time**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**