|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Operating Systems and Compilers** | **Semester:** | **VI** |
| **Date of Performance:** | **28 / 03 / 2025** | **Batch No.:** | **B - 2** |
| **Faculty Name:** | **Prof. Nilesh Lakade** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No.: 7**

**Title: Disk Scheduling Algorithms**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Implementation of Disk Scheduling Algorithm like FCFS, SSTF, SCAN, CSCAN, LOOK.  (any two) |

|  |
| --- |
| **COs to be achieved:** |
| **CO3:** Explain disk organization and file system structure with illustration of disk scheduling algorithms. |

|  |
| --- |
| **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. |
| **Theory** |
| The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.  Goal of Disk Scheduling Algorithm **FCFS Scheduling Algorithm**It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.**SSTF Scheduling Algorithm**Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS. It allows the head to move to the closest track in the service queue. **Scan Algorithm**It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns back and moves in the reverse direction satisfying requests coming in its path. It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back. **Look Scheduling**It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.**:** |

|  |
| --- |
| **Implementation details:** |
| **(printout of code) …any one from FCFS, SSTF**  **Any one from remaining**   1. **FCFS:**   #include <iostream>  #include <vector>  #include <cmath>  using namespace std;  void FCFS(vector<int> requests, int head) {      int totalMovement = 0;      cout << "FCFS Order: " << head;      for (int track : requests) {          totalMovement += abs(track - head);          head = track;          cout << " -> " << track;      }      cout << "\nTotal head movement: " << totalMovement << endl;  }  int main() {      int n, head;      cout << "Enter number of disk requests: ";      cin >> n;      vector<int> requests(n);      cout << "Enter disk track requests: ";      for (int i = 0; i < n; i++)          cin >> requests[i];      cout << "Enter initial head position: ";      cin >> head;      FCFS(requests, head);      return 0;  }     1. **SSTF:**   #include <iostream>  #include <vector>  #include <cmath>  #include <algorithm>  #include <limits>  using namespace std;  void SSTF(vector<int> requests, int head) {      int totalMovement = 0;      vector<bool> visited(requests.size(), false);      cout << "SSTF Order: " << head;      for (int i = 0; i < requests.size(); i++) {          int minDist = numeric\_limits<int>::max();          int minIndex = -1;          for (int j = 0; j < requests.size(); j++) {              if (!visited[j] && abs(requests[j] - head) < minDist) {                  minDist = abs(requests[j] - head);                  minIndex = j;              }          }          if (minIndex == -1) break;          totalMovement += minDist;          head = requests[minIndex];          visited[minIndex] = true;          cout << " -> " << head;      }      cout << "\nTotal head movement: " << totalMovement << endl;  }  int main() {      int n, head;      cout << "Enter number of disk requests: ";      cin >> n;      vector<int> requests(n);      cout << "Enter disk track requests: ";      for (int i = 0; i < n; i++)          cin >> requests[i];      cout << "Enter initial head position: ";      cin >> head;      SSTF(requests, head);      return 0;  } |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **Explain C- Scan algorithm with example.**   The C-SCAN (Circular SCAN) algorithm is a disk scheduling algorithm that moves the disk arm in one direction, servicing all requests until it reaches the last track. Once it reaches the end, it jumps back to the beginning without servicing any requests in between and then continues in the same direction. This ensures that all requests are treated fairly and prevents starvation of requests located at one end of the disk.  For example, consider a disk with track requests at 82, 170, 43, 140, 24, 16, and 190, with an initial head position at 50 and a disk range of 0 to 199. The algorithm sorts the requests and moves in an increasing order, servicing requests at 82, 140, 170, and 190. Once it reaches the highest track, 199, it jumps to track 0 without servicing any requests. It then continues servicing requests at 16, 24, and 43.  The total head movement in this case is calculated as:  (82−50)+(140−82)+(170−140)+(190−170)+(199−190)+(199−0)+(16−0)+(24−16)+(43−24)=391  The C-SCAN algorithm ensures uniform wait times but increases total head movement due to the jump back to the start of the disk.   1. **State the advantages and disadvantages of FCFS algorithm.**   **Advantages:**   * Simple and easy to implement due to its straightforward approach. * Provides fair scheduling as requests are serviced in the order they arrive. * Ensures no starvation since every request is eventually serviced.   **Disadvantages:**   * High seek time as it does not optimize head movement, leading to inefficient scheduling. * Poor performance when the request queue is large, as the head may have to travel long distances unnecessarily. * Not suitable for real-time systems, as urgent requests may have to wait a long time before being serviced. |

|  |
| --- |
| **Conclusion:** |
| In this experiment, we implemented and analyzed various disk scheduling algorithms, including FCFS, SSTF, SCAN, C-SCAN, and C-LOOK. We observed how different algorithms affect disk head movement, response time, and efficiency, with SSTF optimizing seek time, SCAN-based methods ensuring fairness, and FCFS being the simplest but least efficient. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |