Disk I/O Scheduler Simulation Evaluating Performance

Name1: HARSHA VARDHAN(192124074)
Name2: KALYAN. K (192110258)
Name3: SWATHI(192110634)

Disk Scheduling

- Disk scheduling is a technique used by operating systems to manage and prioritize the order of accessing data on disk storage devices.
- The main goal of disk scheduling algorithms is to optimize disk access, reducing disk access time, which includes seek time, rotational latency, and transfer time.
- Disk scheduling algorithms aim to balance between minimizing seek time, reducing disk head movement, and ensuring fair access to the disk among competing processes.
- The choice of disk scheduling algorithm can significantly affect system performance, particularly in environments with heavy disk I/O operations.

Types of Disk Scheduling

- First-Come, First-Served (FCFS):Requests are serviced in the order they arrive. The simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.
- Shortest Seek Time First (SSTF):Selects the request that requires the least movement of the disk arm from its current position. Aims to minimize seek time and can improve performance compared to FCFS.
- SCAN: Also known as the elevator algorithm. The disk arm moves in one direction, servicing requests until the end of the disk, then reverses direction. Prevents long waiting times for requests located at the edge of the disk.

- C-SCAN: Similar to SCAN but the disk arm only moves in one direction, servicing requests until it reaches the end of the disk, then immediately returns to the beginning. Helps prevent large waiting times at the edge of the disk.
- LOOK and C-LOOK: Variants of SCAN and C-SCAN. Similar to their counterparts but only service requests within a certain range of tracks, reducing unnecessary movement of the disk arm.

Functions Used

In a C program implementing disk scheduling, several functions may be used depending on the chosen disk scheduling algorithm and the specific requirements of the program. Here are some common functions that might be used:

- initialize(): This function initializes the disk scheduling algorithm, setting up any necessary data structures or variables.
- add_request(request): This function adds a new disk access request to the scheduling queue. It may involve adding the request to a data structure such as a queue, list, or priority queue.

- get_next_request(): This function retrieves the next request to be serviced according to the disk scheduling algorithm. It typically involves selecting the request based on the algorithm's criteria.
- service_request(request): This function simulates servicing a disk access request. It may involve moving the disk arm, accessing data from the disk, and updating relevant statistics or data structures.
- update_state(): This function updates the internal state of the disk scheduling algorithm after servicing a request. It may involve recalculating priorities, updating positions, or performing other necessary bookkeeping tasks.
- check_for_completion(): This function checks whether all disk access requests have been serviced. It may involve examining the scheduling queue or other data structures to determine if there are any pending requests.

- cleanup(): This function performs any necessary cleanup tasks before exiting the program. It may involve freeing allocated memory, closing files, or releasing other resources.

These functions provide a basic framework for implementing disk scheduling in C. Depending on the complexity of the program and the chosen algorithm, additional functions or variations of these functions may be necessary. Additionally, the specific implementation details will vary based on the requirements and constraints of the system being simulated or controlled.

Working Process of Disk Scheduling Code

- Initialization: The program initializes the disk scheduling algorithm, sets up any necessary data structures, and reads input parameters such as the initial disk head position, the number of disk requests, and the request details (e.g., track numbers).
- Adding Requests: The program adds disk access requests to a scheduling queue or data structure. These requests typically include information such as the track number to be accessed and possibly other attributes like priority.

- Scheduling: The program executes the disk scheduling algorithm to determine the order in which requests should be serviced. This involves selecting the next request to be processed based on the algorithm's criteria, such as minimizing seek time or meeting deadlines.
- Servicing Requests: The program simulates servicing each disk access request according to the scheduling order determined by the algorithm. This involves moving the disk head, accessing data from the disk, and updating relevant statistics or data structures.
- Updating State: After servicing each request, the program updates the internal state of the disk scheduling algorithm as necessary. This may involve recalculating priorities, updating the disk head position, or performing other bookkeeping tasks.

- Completion Check: The program checks whether all disk access requests have been serviced. If there are pending requests, the scheduling process continues; otherwise, the program terminates.
- Cleanup: Finally, the program performs any necessary cleanup tasks before exiting, such as freeing allocated memory, closing files, or releasing other resources.

Conclusion

In the realm of operating systems and storage management, disk scheduling stands as a fundamental mechanism to orchestrate the order of accessing data on disk storage devices, aiming to minimize access time and optimize system performance.

Despite its essential role, disk scheduling algorithms encounter various limitations such as computational overhead, the dominance of seek time in traditional algorithms, and challenges in adapting to modern storage technologies like solid-state drives (SSDs).

These limitations underscore the need for continual research and development in disk scheduling to address evolving computing environments, such as cloud computing and real-time systems, while balancing factors like fairness, scalability, and compatibility with diverse storage architectures.

Thank You