Simulate Your OS: Learn Process Scheduling, Synchronization & More

Lohith Kandibanda
Computer Science and Engineering
Amrita Vishwa Vidyapeetham, Bangalore, Karnataka
bl.en.u4cse22032@students.amrita.edu

Mudumala Varnika Narayani
Computer Science and Engineering
Amrita Vishwa Vidyapeetham, Bangalore, Karnataka
bl.en.u4cse22035@students.amrita.edu

Naga Ruthvika Durupudi
Computer Science and Engineering
Amrita Vishwa Vidyapeetham, Bangalore, Karnataka
bl.en.u4cse22036@students.amrita.edu

Nunnaguppala Rohit
Computer Science and Engineering
Amrita Vishwa Vidyapeetham, Bangalore, Karnataka
bl.en.u4cse22040@students.amrita.edu

Divya K V
Computer Science and Engineering
Amrita Vishwa Vidyapeetham, Bangalore, Karnataka
kv_divya@blr.amrita.edu

***Abstract*—An operating system simulator is a software tool for modeling and analyzing the performance of various operating system components, such as process scheduling, process synchronization, banking algorithms and disk scheduling Algorithm Process scheduling is the process of allocating CPU resources to jobs according to a specific strategy, such as Shortest Job First (SJF), First Come First Served (FCFS) ,Round Robin, Priority Scheduling, and Shortest Remaining Time First (SRJF). Process synchronization ensures simultaneous access to shared resources by multiple processes and solves the producer-consumer problem and the reader-writer problem. Banker's algorithm is used for resource allocation and locking. The simulator simulates distributed conditions and ensures system security .Commonly used disk scheduling algorithms are First Come First Served (FCFS),Shortest Seek Time First (SSTF) , SCAN, C-SCAN, LOOK, and C-LOOK. These algorithms ensure efficient disk access by controlling the order in which block I/O operations are moved to storage volumes. The simulator allows users to observe and learn the complex workings of these operating system processes, allowing them to study and perform experiments in a controlled virtual environment.**

I. Introduction

Operating System is a software layer that facilitates communication between computer hardware and user applications . They are essential for controlling hardware resources, offering a user-friendly interface, and making programme execution easier.An Operating System Simulator is a software tool designed to emulate the behavior and functionality of a operating system in a controlled virtual environment. Without the requirement for real hardware, it enables users to experiment with a number of aspects of operating system function, including Process Scheduling, Process Synchronization, Banker's Algorithm and Disk Scheduling Algorithms.

A. Key Components of an Operating System Simulator

1) Process Scheduling:

- Shortest Job First (SJF) - This scheduling algorithm selects the process with smallest burst time.
- First Come First Served (FCFS) - This scheduling algorithm executes in the order they arrive in the ready queue.
- Round Robin - This scheduling algorithm by executing each process for a small unit of time (quantum or time slice), and moving to the next process in a circular queue.
- Priority Scheduling - In this scheduling algorithm processes are assigned with priorities and the scheduling algorithm executes the process with highest priority and the moves to the next process.
- Shortest Remaining Time First (SRTF)- this scheduling algorithm is non preemptive scheduling algorithm executes the process with the shortest remaining burst time.

2) Process Synchronization:

- Producer-Consumer Problem - The bounded buffer problem, also recognized as the producer-consumer problem, entails the coordination of two distinct processes: the producer, responsible for generating new data and depositing them into a fixed-size buffer, and the consumer, tasked with retrieving data from the buffer for processing. This problem necessitates an organization where producers can continue producing new data without interruption when the buffer reaches its maximum capacity, while consumers are prevented from consuming data when the buffer is empty. Conversely, when the buffer becomes empty, consumers remain in a state of waiting until signaled by producers before proceeding with any further actions.
- Reader-Writer Problem - The readers-writers problem arises from the synchronization of processes accessing a shared object, where readers exclusively read data and writers solely perform write operations. This scenario

necessitates preventing data corruption caused by concurrent write operations or conflicts between reads and writes. To address concurrency issues, synchronization mechanisms must be implemented to regulate access and maintain data consistency across multiple processes.

3) Banker's Algorithm:
The Banker's algorithm is a resource allocation and deadlock avoidance technique used to manage resources efficiently within a system.

4) Disk Scheduling Algorithms:

- First Come First Served (FCFS) - The (FCFS) first come first served disk scheduling algorithm adds requests that come in order one after the other to the tail of the queue and the request that comes next in line is the one at the head of the queue. Although this method is easy to implement, it may not always generate the least head movement which is the best choice. Given two requests, the number of head movements between them can be calculated by finding out how many tracks are traversed.
- Shortest Seek Time First (SSTF) - Shortest seek time first Algorithm selects the disk I/O request which needs to move the least number of disk arms from its present position in any direction. It reduces the total seek time as compared to FCFS. Algorithm allows the head to move to the closest track in the service queue.
- SCAN - ”In SCAN, the disk arm moves in a specific direction, servicing all requests encountered on its path until it reaches the end of the disk. Then it turns in the opposite direction and travels back, fulfilling requests encountered in the reverse order. This is analogous to the elevator, so the algorithm is popularly called the elevator algorithm. SCAN optimizes disk access because it effectively services requests in a sequential manner, saving on seek time and enhancing performance overall.”
- C-SCAN - The C-SCAN algorithm has the disk arm move in a certain direction and service requests until it reaches the last cylinder. After reaching the last cylinder, it switches to the last cylinder of the opposite direction and does not service any requests. In a change of direction, it will then start to move, service the remaining requests, and so on. This is in a way to maximize the access of disks by eliminating as many head movements as possible and maximizing the overall performance of the disk.
- LOOK - ”The LOOK disk scheduling algorithm is very similar to the SCAN algorithm except that the disk arm stops its inward or outward movement once no more requests are in that direction. In other words, this is where the difference between SCAN lies, which has the disk arm continuing its movement in one direction to the end, even if no more requests are there. LOOK minimizes the waste of SCAN by doing the maximum amount of disk access and minimizing unnecessary head movement.”
- C-LOOK - The C-Look disk scheduling algorithm is similar in nature to the C-SCAN algorithm. It travels outwards until the highest cylinder of the requests is reached, jumps over to the lowest cylinder of requests without serving them, and continues the outward journey to service the remaining requests. This means the algorithm is going to allocate access to the disk for the optimal number of requests by minimizing head movements as much as possible and reaching full disk performance.

II. Literature Survey

Real-time systems rely on efficient scheduling algorithms and operating system support to meet strict timing constraints. Various paradigms exist for scheduling approaches, including static table-driven, static priority preemptive, dynamic planning-based, and dynamic best effort scheduling. Novel distributed resource-scheduling algorithms have been developed to facilitate real-time operations on replicated distributed files, enhancing efficiency and minimizing message overhead. Priority-based preemptive scheduling, such as Earliest Deadline First (EDF) and Ant Colony Optimization Based (ACO) schedulers, are commonly used in real-time systems to ensure high-priority task execution within specified timeframes. Additionally, Support Vector Machine (SVM) learning methods have been proposed to predict priorities of multiple queues in real-time operating systems, enabling effective handling of nonhomogeneous tasks with high performance.

A comparative analysis of CPU scheduling algorithms is crucial for enhancing Central Processing Unit efficiency. Various studies have focused on comparing different scheduling algorithms to determine the most efficient one. Traditional real-time scheduling algorithms aim to meet requirements for rapid execution, high performance, and low power consumption. These algorithms play a significant role in various applications such as manufacturing, data processing systems, and aviation. Researchers have explored multiple algorithms like First Come First Serve, Priority Scheduling, Shortest Job First, and Round Robin to assess their impact on CPU performance. Through simulation and performance analysis, new scheduling algorithms have been proposed to optimize CPU utilization, response time, and overall system efficiency
.

The integration of round-robin and priority scheduling algorithms in CPU scheduling aims to enhance real-time operating system performance by addressing issues like starvation. This new approach assigns both time quantum and priority index to processes, effectively reducing waiting time, response time, and turnaround time compared to traditional round-robin algorithms. By implementing aging and assigning new priorities, the proposed algorithm mitigates the drawbacks of existing round-robin methods, which struggle with high context switch rates and low throughput in real-time systems. Comparative analyses demonstrate the superiority of the new algorithm in terms of average waiting time, average turnaround time, and number of context switches, showcasing its potential to eliminate starvation and enhance CPU performance.

A continuum of disk scheduling algorithms, ranging from SSTF to SCAN, has been defined to optimize performance in serving both continuous and discrete data workloads efficiently. These algorithms aim to minimize response times for discrete requests while ensuring uninterrupted delivery of continuous data, crucial for multimedia applications. By dynamically adapting to workload variations and providing adaptive admission control, these algorithms enhance resource utilization and support more clients simultaneously, even under heavy traffic conditions . The research emphasizes the need for efficient disk I/O scheduling to address the increasing gap between processor speed and disk speed, highlighting the importance of developing algorithms with provable performance guarantees . Overall, these scheduling strategies offer significant improvements in response times, fairness, and continuous data delivery, showcasing their potential for enhancing multimedia server performance .

A continuum of disk scheduling algorithms, ranging from SSTF to SCAN, has been defined to optimize performance in serving both continuous and discrete data workloads efficiently. These algorithms aim to minimize response times for discrete requests while ensuring uninterrupted delivery of continuous data, crucial for multimedia applications. By dynamically adapting to workload variations and providing adaptive admission control, these algorithms enhance resource utilization and support more clients simultaneously, even under heavy traffic conditions . The research emphasizes the need for efficient disk I/O scheduling to address the increasing gap between processor speed and disk speed, highlighting the importance of developing algorithms with provable performance guarantees . Overall, these scheduling strategies offer significant improvements in response times, fairness, and continuous data delivery, showcasing their potential for enhancing multimedia server performance .

III. METHODOLOGY

Choosing the optimal scheduling algorithm is vital for maximizing performance and ensuring fairness among competing processes. This comprehensive methodology empowers you to delve into the intricacies of various scheduling algorithms, providing a clear roadmap to analyze their effectiveness.

Workflow:

1) Process scheduling: New programs, called processes, constantly enter the system. These processes don't get to execute immediately. Instead, they join a special queue known as the ”ready queue.” This queue essentially holds processes waiting for their turn on the CPU, the central processing unit executes instructions. The moment a process enters the ready queue signifies its arrival time. The operating system, employs a crucial component called the process scheduler. This scheduler is responsible for the critical decision of selecting the next process to use the CPU. Different scheduling algorithms exist, like FCFS (First Come First Served), SJF (Shortest Job First), or Priority scheduling. Once chosen by the scheduler, the process is granted access to the CPU. This is when the process starts executing its instructions. The duration it occupies the CPU for this purpose is known as the process execution time. When a process finishes its execution, meaning it completes all its tasks, it exits the CPU stage. This moment marks the process completion time. If another process is eagerly waiting in the ready queue and qualifies for execution based on the chosen algorithm, the operating system performs a context switch. This involves a quick but necessary step: saving the state of the current process (like its registers and memory pointers) and loading the state of the new process, essentially preparing it for its turn on the CPU. Context switching comes with a minor overhead, a small delay, due to the saving and loading involved. Processes that aren't chosen for immediate execution by the scheduler have to wait in the ready queue. The waiting time refers to the total duration a process spends in this queue before it gets its shot at the CPU. These steps become a continuous loop until all processes in the system have completed their execution. This ensures that each program gets a fair share of CPU time, allowing the computer system to operate efficiently.

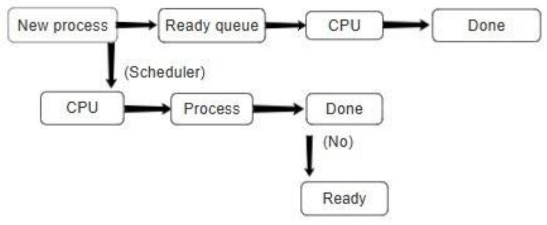


Fig. 1: Process Scheduling

2) Process synchronization: Processes often reach a specific point in their execution where they need to interact with a shared resource. This crucial point is called the ”synchronization point.” When a process needs to access the shared resource, it enters a special code segment called the ”critical section.” This ensures exclusive access and prevents data conflicts. Process synchronization mechanisms employ a concept called ”mutual exclusion.” This term essentially means that only one process can occupy the critical section at any given moment. Other processes attempting to enter are politely (or sometimes forcefully) blocked until the current occupant finishes their business. Once inside the critical section, the process can safely access and modify the shared resource. No other process can interfere, ensuring data integrity. When the process finishes using the shared resource, it exits the critical section. This opens the door for the next waiting process, allowing it a chance to enter. The process synchronization mechanism might involve releasing any locks or signals it held. This informs other processes that the resource is no longer in use and they can try to access it. The blocked processes waiting to access the shared resource are now eligible to proceed. They can attempt to enter the critical section based on the specific rules defined by the chosen synchronization mechanism.

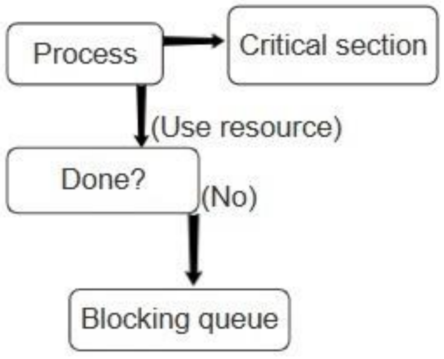


Fig. 2: Process Scheduling

3) Reader-writer problem: A reader process arrives, eager to access the shared resource for reading. Before diving in, the reader checks two things: Is any writer currently occupying the ”critical section,” meaning they’re exclusively modifying the resource? and there already other readers present? If the coast is clear (no writers present and the number of readers is below a predefined limit to allow concurrent reading), the reader gets the green light to proceed. The reader increments a counter variable, essentially keeping track of how many readers are currently accessing the resource. Now, the reader can safely access and read the shared resource without disruption from other processes, ensuring they get a consistent view of the data. Once finished, the reader decrements the reader count variable, indicating one less reader is using the resource. A writer process arrives, needing to modify the shared resource. Similar to the reader, the writer checks for any ongoing activity: Are any readers or writers currently accessing the resource? If the document is free (no readers or writers present), the writer gets access by entering the critical section. To prevent interruptions, the writer acquires a lock (like a semaphore or mutex) on the resource. This essentially signals to other processes (readers or writers) that the resource is unavailable for modification. With exclusive access secured, the writer can freely modify the shared resource as needed. Once finished writing, the writer releases the lock, allowing other processes to access the resource again.

4) Bankers algorithm: This is like the bank’s current stock of cash (resources) for each type (e.g., memory, CPU cycles). It’s represented as a list showing how many units of each resource are currently free. Think of this as a customer’s credit limit. It’s a two-dimensional table (matrix) where Max[i][j] tells us the maximum number of resource units of type j that a process Pi can ever request. This is similar to a customer’s outstanding loan amount. It’s another two-dimensional matrix where Allocation[i][j] represents the number of resource units of type j that are currently allocated to process Pi. When a process needs additional resources, it submits a request. The Banker’s Algorithm doesn’t blindly grant the request. Instead, it simulates the allocation to check if it’s safe. The algorithm temporarily adjusts the Available list, subtracting the requested resources to see if there would be enough left for others. It also modifies the Allocation matrix, adding the requested resources to the process’s current allocation, simulating the loan being granted. A safe sequence is like an order in which all customers (processes) can finish their transactions (executions) without encountering a deadlock. It searches for a process that can complete its execution based on the Max and Allocation matrices. This means the process has enough resources (like having enough money to complete a transaction). If such a process is found, it’s like a customer finishing their transaction and returning the loaned resources (allocation). The algorithm simulates this by permanently removing the process’s used resources from the Available list and taking it out of the running. These steps are repeated until either all processes can be finished in a specific order (safe sequence) or no process can finish due to insufficient resources. If a safe sequence is found, it means granting the requested resources wouldn’t lead to a deadlock. The process gets the resources it needs (loan is approved). The temporary adjustments made to Available and Allocation become permanent, reflecting the actual allocation. If no safe sequence exists, the request is denied to prevent a potential deadlock. The temporary adjustments are discarded, essentially saying ”not this time” to the loan request.

5) Disk scheduling algorithms:

FCFS: The moment a process enters the queue signifies its arrival time. The operating system employs a process scheduler, (decides which process gets the CPU). The FCFS algorithm dictates a simple rule: the process that arrived first gets served first. Once chosen by the scheduler, the process gets allocated the CPU and starts executing its instructions. This is similar to the customer at the front of the line receiving their coffee order. The duration the process occupies the CPU for this purpose is known as the process execution time. When a process finishes its execution, it exits the CPU stage. This moment marks the process completion time. If another process is eagerly waiting in the queue, it becomes eligible for CPU allocation as soon as the current process finishes. This ensures a smooth transition and keeps the CPU busy. These steps are continuously repeated until all processes in the system have completed their execution. This ensures that each program gets a fair share of CPU time, allowing the computer system to operate efficiently.

SSTF: New requests (processes) arrive, each specifying the track they need to access. These requests form a queue, waiting for the head to service them. The disk head has a current position on a specific track. This position is crucial for determining the seek time, which is the time it takes for the head to move from its current track to the requested track. The SSTF algorithm prioritizes requests based on their seek time. Unlike FCFS (First Come, First Served), it doesn't simply choose the request at the front of the queue. Instead, it identifies the request in the queue that requires the shortest seek time from the head's current position. This closest neighbor gets serviced first. Once the closest request is chosen, the head moves to the requested track and performs the read or write operation. After servicing the request, the head's current position is updated to the newly accessed track. The served request is removed from the queue. The steps are repeated until all requests in the queue have been serviced. The algorithm continuously seeks the closest remaining request, minimizing the total head movement and reducing seek time. In some cases, there might be multiple requests with the same minimal seek time from the head's current position. The SSTF algorithm might have a tie-breaking mechanism to decide which one to service first (e.g., favoring requests that arrived earlier).

SCAN: Similar to SSTF, requests for accessing different disk tracks are stored in a queue, waiting for the disk head to service them. The disk head starts at a specific track on the disk platter. This initial position plays a crucial role in determining the direction of head movement. Unlike SSTF, SCAN doesn't prioritize based on individual seek times. Instead, it scans the disk in a chosen direction. There are two variations: SCAN (Left to Right): The head starts at its initial position and moves towards one end of the disk platter (usually the rightmost end), servicing all requests it encounters along the way. SCAN (Right to Left): The head starts at its initial position and moves towards the opposite end of the disk platter (usually the leftmost end), servicing requests as it encounters them. Once the head reaches the designated end of the disk (rightmost or leftmost depending on the chosen direction), it reverses its direction. The head starts scanning in the opposite direction, servicing any remaining requests that it skipped during the initial scan. These steps are repeated until all requests in the queue have been serviced. This ensures that all parts of the disk platter are eventually accessed, catering to requests scattered across different tracks.

C-SCAN: Similar to SCAN, C-SCAN relies on a queue to store requests for accessing different disk tracks. The disk head starts at a specific track on the disk platter, just like in SCAN. C-SCAN introduces a key difference. It treats the disk platter as a circular track. Imagine wrapping the outermost track (end) to the beginning, creating a continuous loop. This eliminates the need for a turnaround at the physical ends of the disk. C-SCAN also chooses a servicing direction (left or right) like SCAN. It starts moving the head in that direction, servicing all requests it encounters along the way. However, unlike SCAN which physically reaches the end of the disk, C-SCAN reaches a logical ”end” when it encounters the highest or lowest track number (depending on the chosen direction) in the request queue. Instead of reversing direction, C-SCAN exploits the circular concept. It immediately jumps (seeks) to the opposite end of the disk platter (the lowest or highest track number, depending on the direction). This jump utilizes the idle time when the head would otherwise be traveling back in the opposite direction. Once at the other end, C-SCAN starts servicing any remaining requests in the queue, working its way back towards the starting position it encountered earlier. These steps are repeated until all requests in the queue have been serviced. The circular approach ensures all requests are eventually accessed, regardless of their initial location.

LOOK: Similar to SCAN and C-SCAN, LOOK relies on a queue to store requests for accessing different disk tracks. The disk head starts at a specific track on the disk platter, just like in SCAN and C-SCAN. LOOK also chooses a servicing direction (left or right) like SCAN. It starts moving the head in that direction, servicing all requests it encounters along the way. Unlike SCAN which physically reaches the end of the disk and C-SCAN which reaches a logical end based on the request queue, LOOK stops servicing requests when it reaches the last request in the chosen direction (furthest request in that direction). The key difference between LOOK and SCAN/C-SCAN lies here. Instead of completely reversing direction, LOOK only reverses as far as the first request in the opposite direction (closest request in that direction) that wasn't serviced yet. Once positioned at the first unserviced request in the opposite direction, LOOK starts servicing remaining requests, working its way back towards the starting position. These steps are repeated until all requests in the queue have been serviced. LOOK ensures all requests are eventually accessed, regardless of their initial location.

C-LOOK: Similar to SCAN and LOOK, C-LOOK relies on a queue to store requests for accessing different disk tracks. The disk head starts at a specific track on the disk platter, just like in other SCAN algorithms. C-LOOK, like C-SCAN, treats the disk platter as a circular track. Imagine wrapping the outermost track (end) to the beginning, creating a continuous loop. This eliminates the need for physical turnaround at the disk's ends. C-LOOK chooses a servicing direction (left or right) like SCAN and LOOK. It starts moving the head in that direction, servicing all requests it encounters along the way. However, unlike SCAN which physically reaches the end of the disk and LOOK which stops at the last request, C-LOOK reaches a logical ”end” when it encounters the highest or lowest track number (depending on the chosen direction) in the entire request queue, not just the currently serviced direction. This includes requests beyond the head's current position. Instead of reversing direction, C-LOOK exploits the circular concept. It immediately jumps (seeks) to the opposite end of the disk platter (the lowest or highest track number, depending on the direction). This jump utilizes the idle time when the head would otherwise be traveling back in the opposite direction. Once at the other end, C-LOOK starts servicing all remaining requests in the queue. This includes requests encountered during the initial scan and any requests beyond the head's starting position that were included in the logical "end" determination. C-SCAN only serviced remaining requests encountered during the initial scan. These steps are repeated until all requests in the queue have been serviced. The circular approach ensures all requests are eventually accessed, regardless of their initial location.

IV. Comparative Analysis

A. Process Scheduling

1) Shortest Job First (SJF):

Advantages:

- Reduces average waiting time by prioritizing small processes.
- Proper exploitation of CPU resources.

Disadvantages:

- It needs knowledge about process burst times, which are not always available.
- May reduce the performance of long processes if short processes always enter the system.

2) First Come First Served (FCFS):

Advantages:

- Straightforward and easy to put into play.
- The fairness for resource allocation of long-running procedures.

Disadvantages:

- Long average waiting time, especially for long processes, is one of the main reasons for low performance.
- Subjected to the time lag effect, in which shorter processes are delayed behind longer processes.

3) Priority Scheduling:

Advantages:

- This makes it possible to focus on the vital processes first.
- Enables processing of time-critical tasks that are on the forefront.

Disadvantages:

- Possibility of starvation to the low priority tasks.
- Can result in priority inversion and deadlock in some particular cases.

4) Shortest Remaining Time First (SRTF):

Advantages:

- Decreases average waiting time by preemptively scheduling the shortest job currently in the process.
- Fast for the operating systems that have randomly changing process sizes.

Disadvantages:

- The option of starvation due to the extended duration of processes.
- Increased overhead due to the occurrence of context switching more frequently.

B. Process Synchronization

1) Producer-Consumer and Reader-Writer scenarios:

Advantages:

- Effective interaction and communication among the processes.
- Data corruption and race condition prevention.

Disadvantages:

- Risks of getting into a deadlock or livelock condition.
- The complexity of the synchronization mechanisms is increasing.

C. Banker's Algorithm

Advantages:

- Makes sure safe resource allocation and eliminates deadlocks.
- Provides a tool for managing resources in the most efficient way.

Disadvantages:

- Knowledge about availability of future resource requirements that is not always accessible is needed.
- Complexity emerges as the number of processes and resources grow.

D. Disk Scheduling Algorithms

1) First Come First Served (FCFS):

Advantages:

- Simple and easy to implement.
- Fairness in disk access.

Disadvantages:

- Congestion effect during periods of high demand.

2) Shortest Seek Time First (SSTF):

Advantages:

- Reduction in seek time.
- Better performance rates.

Disadvantages:

- Starvation may occur among distant requests.
- Overhead for frequent reordering of requests.

3) SCAN:

Advantages:

- Justice can be weighed against efficiency.
- Eliminates queue starvation of requests at the end of the disk.

Disadvantages:

- Higher average seek time than SSTF.
- Higher complexity in implementation.

4) C-SCAN:

Advantages:

- Performance improvement by serving requests in one direction.
- Prevents starvation of requests.

Disadvantages:

- Increased average seek times relative to LOOK.
- Limited availability of agents may cause trouble for customers.

5) LOOK:

Advantages:

- Reduces the sum of seek time.
- Better performance than SCAN.

Disadvantages:

- May result in less even seek time compared to SSTF.
- Restriction of equality in borrower servicing.

6) C-LOOK:

Advantages:

- Identifies and resolves requests within a specified range.
- Prevents starvation of requests.

Disadvantages:

- Increased average seek times relative to LOOK.
- Customers may face trouble due to limited availability of agents.

V. Results

An Operating System Simulator provides knowledge of operating systems concepts like Process Scheduling, Process synchronization, Banking Algorithms, and Disk scheduling Algorithm.

Process Scheduling provides comprehensive insights into the performance of each process scheduling algorithm, including their execution sequences, average waiting times, average turnaround times, and Gantt chart.

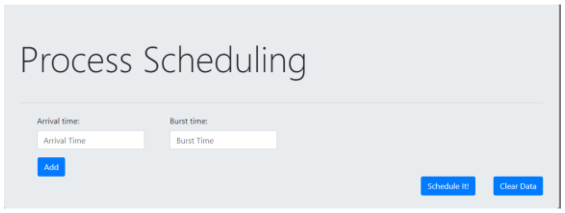


Fig. 3: Process Scheduling Simulator

Shortest Job First (SJF):

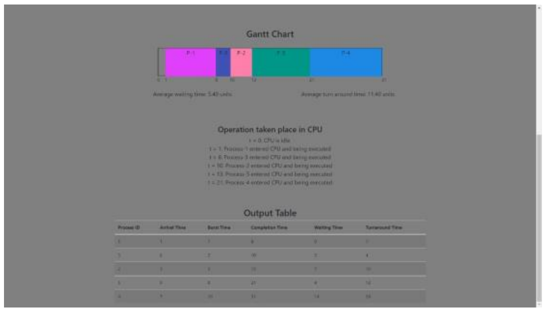
Example 1: There are five Processes named as P1, P2, P3, P4, and P5. Their arrival time and burst time are given as follows:

Average Waiting Time: 5.40 units

Average Turnaround Time: 11.40 units

TABLE I: Process Information for SJF

Sr. No.	Arrival Time	Burst Time
1	1	7
2	3	3
3	6	2
4	7	10
5	9	8



Average waiting time: 5.40 units

Average turn around time: 11.40 units

Operation taken place in CPU

t = 0: CPU is idle

t = 1: Process-1 entered CPU and being executed

t = 8: Process-3 entered CPU and being executed

t = 10: Process-2 entered CPU and being executed

t = 13: Process-5 entered CPU and being executed

t = 21: Process-4 entered CPU and being executed

Output Table

Process ID	Arrival Time	Burst Time	Completion Time	Waiting Time	Turnaround Time
1	1	7	8	0	7
3	6	2	10	2	4
2	3	3	13	7	10
5	9	8	21	4	12
4	7	10	31	14	24

Fig. 4: Process Scheduling SJF Algorithm

and P5. Their arrival time and burst time are given as follows:

Average Waiting Time: 3.20 units

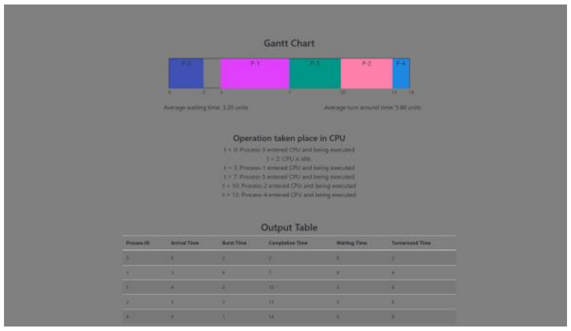
Average Turnaround Time: 5.80 units

First Come First Serve (FCFS):

Example 2: There are five Processes named as P1, P2, P3, P4,

TABLE II: Process Information for FCFS

Sr. No.	Arrival Time	Burst Time
1	3	4
2	5	3
3	0	2
4	5	1
5	4	3



Average waiting time: 3.20 units

Average turn around time: 5.80 units

Operation taken place in CPU

t = 0: Process-3 entered CPU and being executed

t = 2: CPU is idle

t = 3: Process-1 entered CPU and being executed

t = 7: Process-5 entered CPU and being executed

t = 10: Process-2 entered CPU and being executed

t = 13: Process-4 entered CPU and being executed

Output Table

Process ID	Arrival Time	Burst Time	Completion Time	Waiting Time	Turnaround Time
3	0	2	2	0	2
1	3	4	7	0	4
5	4	3	10	3	6
2	5	3	13	5	8
4	5	1	14	8	9

Fig. 5: Process Scheduling FCFS Algorithm

Round Robin Algorithm:

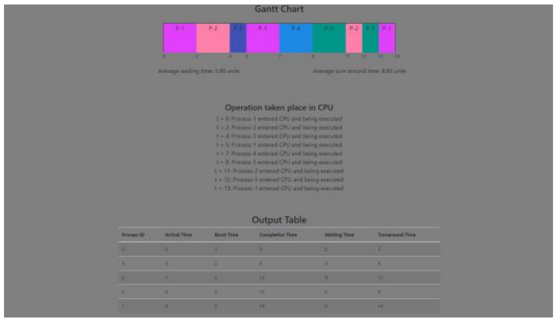
Example 3: There are five Processes named as P1, P2, P3, P4, and P5. Their arrival time and burst time are given as follows:

If the CPU scheduling policy is Round Robin with a time quantum = 2 units, calculate the average waiting time and average turnaround time.

Average Waiting Time: 5.80 units

TABLE III: Process Information for Round Robin

Sr. No.	Arrival Time	Burst Time
1	0	5
2	1	3
3	2	1
4	3	2
5	4	3



Average waiting time: 5.80 units   Average turn around time: 8.60 units

Operation taken place in CPU

```
t = 0: Process-1 entered CPU and being executed
t = 2: Process-2 entered CPU and being executed
t = 4: Process-3 entered CPU and being executed
t = 5: Process-1 entered CPU and being executed
t = 7: Process-4 entered CPU and being executed
t = 9: Process-5 entered CPU and being executed
t = 11: Process-2 entered CPU and being executed
t = 12: Process-5 entered CPU and being executed
t = 13: Process-1 entered CPU and being executed
```

Output Table

Process ID	Arrival Time	Burst Time	Completion Time	Waiting Time	Turnaround Time
3	2	1	5	2	3
4	3	2	9	4	6
2	1	3	12	8	11
5	4	3	13	6	9
1	0	5	14	9	14

Fig. 6: Process Scheduling Round Robin Algorithm

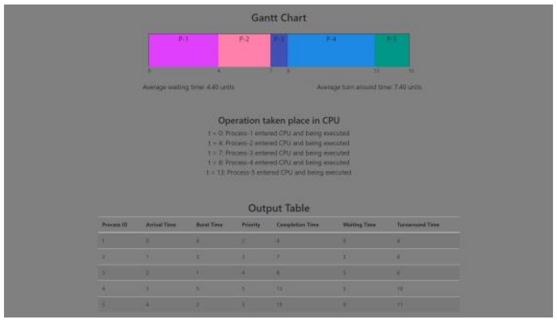
Average Turnaround Time: 8.60 units

Priority Scheduling Algorithm:
Example 4: There are five Processes named as P1, P2, P3, P4, and P5. Their arrival time, burst time, and priority are given as follows:

TABLE IV: Process Information for Priority Scheduling

Sr. No.	Arrival Time	Burst Time	Priority
1	0	4	2
2	1	3	3
3	2	1	4
4	3	5	5
5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turnaround time. (Higher number represents higher priority)



Average waiting time: 4.40 units   Average turn around time: 7.40 units

Operation taken place in CPU

```
t = 0: Process-1 entered CPU and being executed
t = 4: Process-2 entered CPU and being executed
t = 7: Process-3 entered CPU and being executed
t = 8: Process-4 entered CPU and being executed
t = 13: Process-5 entered CPU and being executed
```

Output Table

Process ID	Arrival Time	Burst Time	Priority	Completion Time	Waiting Time	Turnaround Time
1	0	4	2	4	0	4
2	1	3	3	7	3	6
3	2	1	4	8	5	6
4	3	5	5	13	5	10
5	4	2	5	15	9	11

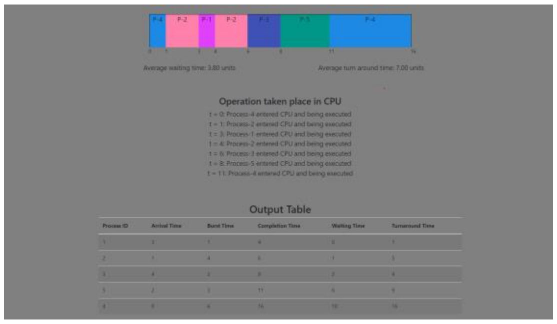
Fig. 7: Process Scheduling Priority Scheduling Algorithm

Average Waiting Time: 4.40 units
Average Turnaround Time: 7.40 units

Shortest Remaining Time First (SRTF):
Example 5: There are five Processes named as P1, P2, P3, P4, and P5. Their arrival time and burst time are given as follows:

TABLE V: Process Information for SRTF

Sr. No.	Arrival Time	Burst Time
1	3	1
2	1	4
3	4	2
4	0	6
5	2	3



Average waiting time: 3.80 units   Average turn around time: 7.00 units

Operation taken place in CPU

```
t = 0: Process-4 entered CPU and being executed
t = 1: Process-2 entered CPU and being executed
t = 3: Process-1 entered CPU and being executed
t = 4: Process-2 entered CPU and being executed
t = 6: Process-3 entered CPU and being executed
t = 8: Process-5 entered CPU and being executed
t = 11: Process-4 entered CPU and being executed
```

Output Table

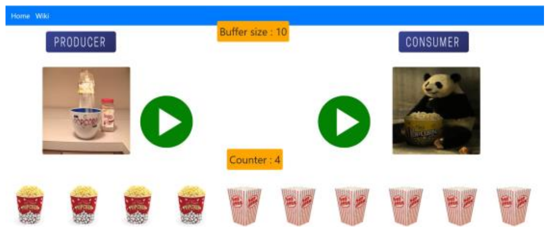
Process ID	Arrival Time	Burst Time	Completion Time	Waiting Time	Turnaround Time
1	3	1	4	0	1
2	1	4	6	1	5
3	4	2	8	2	4
5	2	3	11	6	9
4	0	6	16	10	16

Fig. 8: Process Scheduling SRTF Algorithm

Average Waiting Time: 3.80 units
Average Turnaround Time: 7.00 units

Process Synchronization provides a comprehensive overview of the analysis and implementation, behavior, and performance of various synchronization techniques such as semaphores, mutex locks, and condition variables for Producer-Consumer Problem and Reader-Writer Problem.

Producer Consumer Problem :
Example 6: In this scenario, a popcorn production and consumption system is simulated, where a producer produces popcorn and places it in a buffer of fixed size. The consumer retrives the popcorn from the buffer and consumes it. The buffer is limited in size and the producer must wait when the buffer is full, the consumer must wait when the buffer is empty.



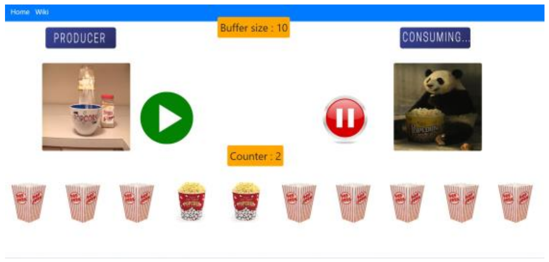
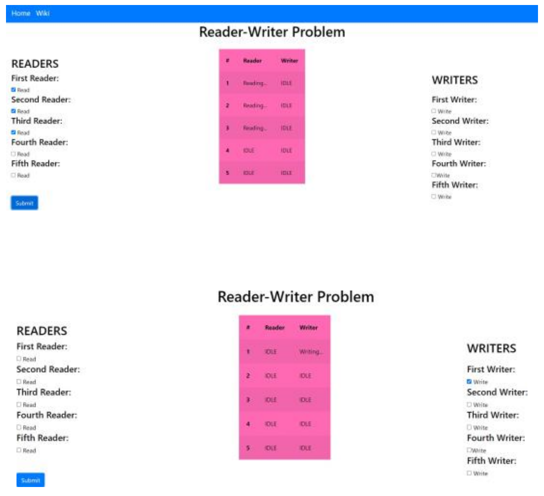


Fig. 9: Producer Consumer Problem

Reader Writer Problem:

Example 7: The readers-writers problem is used to manage synchronization so that there are no problems with the object data. In this scenario, if two readers access the object at the same time there is no problem. However, if two writers or a reader and writer access the object at the same time, there may be problems.



Reader-Writer Problem

READERS

First Reader: [✓ Read]
Second Reader: [✓ Read]
Third Reader: [✓ Read]
Fourth Reader: ☐ Read
Fifth Reader: ☐ Read

#	Reader	Writer
1	Reading...	IDLE
2	Reading...	IDLE
3	Reading...	IDLE
4	IDLE	IDLE
5	IDLE	IDLE

WRITERS

First Writer: ☐ Write
Second Writer: ☐ Write
Third Writer: ☐ Write
Fourth Writer: ☐ Write
Fifth Writer: ☐ Write

Submit

Reader-Writer Problem

READERS

First Reader: ☐ Read
Second Reader: ☐ Read
Third Reader: ☐ Read
Fourth Reader: ☐ Read
Fifth Reader: ☐ Read

#	Reader	Writer
1	IDLE	Writing...
2	IDLE	IDLE
3	IDLE	IDLE
4	IDLE	IDLE
5	IDLE	IDLE

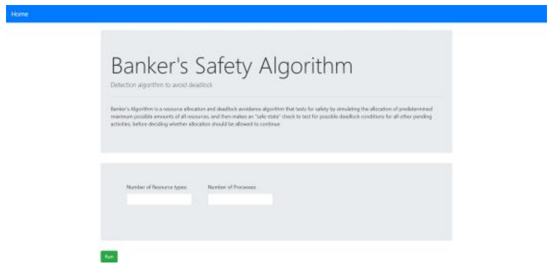
WRITERS

First Writer: [✓ Write]
Second Writer: ☐ Write
Third Writer: ☐ Write
Fourth Writer: ☐ Write
Fifth Writer: ☐ Write

Submit

Fig. 10: Reader Writer Problem

Banker Algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an" safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to con tinue.



Banker's Safety Algorithm

Detection algorithm to avoid deadlock

Banker's Algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

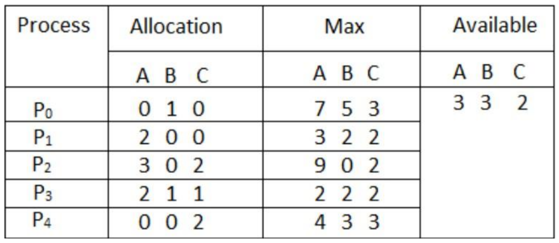
Number of Resource types:

Number of Processes:

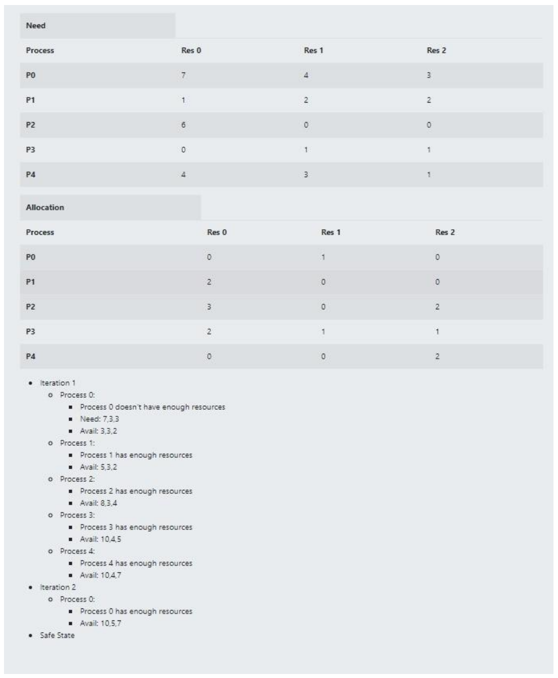
Run

Fig. 11: Banker Algorithm

Example 8: Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. The following are the resource types: A has 10, B has 5 and the resource type C has 7 instances.



Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	



Need

Process	Res 0	Res 1	Res 2
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Allocation

Process	Res 0	Res 1	Res 2
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

- Iteration 1
  - Process 0:
    - Process 0 doesn't have enough resources
    - Need: 7,3,3
    - Avail: 3,3,2
  - Process 1:
    - Process 1 has enough resources
    - Avail: 5,3,2
  - Process 2:
    - Process 2 has enough resources
    - Avail: 8,3,4
  - Process 3:
    - Process 3 has enough resources
    - Avail: 10,4,5
  - Process 4:
    - Process 4 has enough resources
    - Avail: 10,4,7
- Iteration 2
  - Process 0:
    - Process 0 has enough resources
    - Avail: 10,5,7
- Safe State

Fig. 12: Banker Algorithm

Disk Scheduling Algorithm is the method that computer operating systems use to decide in which order the block I/O operations will be submitted to storage volumes. I/O scheduling is sometimes called disk scheduling

In this exercise, we'll explore various disk scheduling algorithms including FCFS (First-Come, First-Served), SSTF (Shortest Seek Time First), SCAN, C-SCAN, LOOK, and C-LOOK. Each algorithm aims to optimize the movement of the disk arm to minimize the seek time.

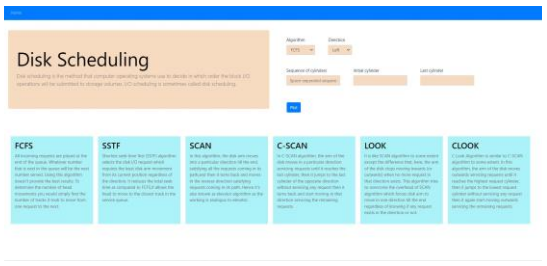
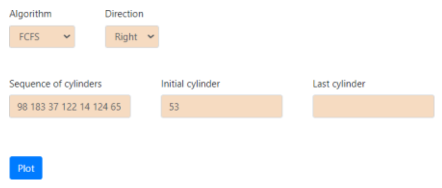


Fig. 13: Disk Scheduling Algorithm

FCFS (First-Come, First-Served)

Example 9: Consider a disk with 1000 cylinders numbered from 0 to 999. The requests for disk I/O are as follows: 98, 183, 37, 122, 14, 124, 65, 67. The initial position of the disk arm is at cylinder 53. The Direction is Right. Implement the FCFS disk scheduling algorithm and calculate the total seek time. Also, plot the sequence of cylinder accesses on a graph.



Algorithm: FCFS
Direction: Right
Sequence of cylinders: 98 183 37 122 14 124 65
Initial cylinder: 53
Last cylinder:

Fig. 14: First Come First Serve Algorithm

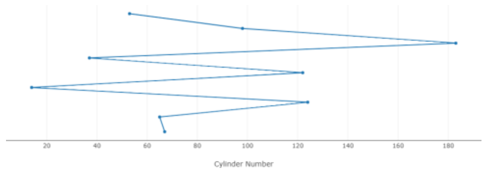


Fig. 15: First Come First Serve Algorithm Graph

Absolute difference between 98 and 53: $|98 - 53| = 45$

Absolute difference between 183 and 98: $|183 - 98| = 85$

Absolute difference between 37 and 183: $|37 - 183| = 146$

Absolute difference between 122 and 37: $|122 - 37| = 85$

Absolute difference between 14 and 122: $|14 - 122| = 108$

Absolute difference between 124 and 14: $|124 - 14| = 110$

Absolute difference between 65 and 124: $|65 - 124| = 59$

Absolute difference between 67 and 65: $|67 - 65| = 2$

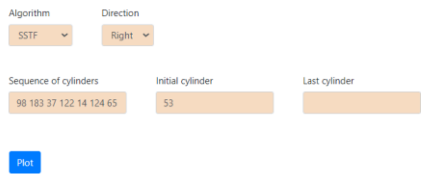
Now, we sum up these seek times:

$45 + 85 + 146 + 85 + 108 + 110 + 59 + 2 = 640$

So, the total seek time is 640.

SSTF (Shortest Seek Time First)

Example 10: Consider a disk with 1000 cylinders numbered from 0 to 999. The requests for disk I/O are as follows: 98, 183, 37, 122, 14, 124, 65, 67. The initial position of the disk arm is at cylinder 53. The Direction is Right .Implement the SSTF disk scheduling algorithm and calculate the total seek time. Also, plot the sequence of cylinder accesses on a graph.



Algorithm: SSTF
Direction: Right
Sequence of cylinders: 98 183 37 122 14 124 65
Initial cylinder: 53
Last cylinder:

Fig. 16: Shortest Seek Time First Algorithm

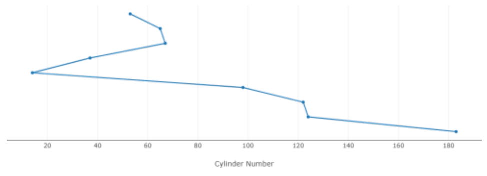


Fig. 17: Shortest Seek Time First Algorithm Graph

Seek from cylinder 53 to cylinder 65: $|53 - 65| = 12$
Seek from cylinder 65 to cylinder 67: $|65 - 67| = 2$
Seek from cylinder 67 to cylinder 37: $|67 - 37| = 30$
Seek from cylinder 37 to cylinder 14: $|37 - 14| = 23$
Seek from cylinder 14 to cylinder 98: $|14 - 98| = 84$
Seek from cylinder 98 to cylinder 122: $|98 - 122| = 24$
Seek from cylinder 122 to cylinder 124: $|122 - 124| = 2$
Seek from cylinder 124 to cylinder 183: $|124 - 183| = 59$
Now, we sum up these seek times:
$12 + 2 + 30 + 23 + 84 + 24 + 2 + 59 = 236$
So, the total seek time is 236.

SCAN

Example 11: Consider a disk with 100 cylinders numbered from 0 to 99. The requests for disk I/O are as follows:98, 137, 122, 183, 14, 133, 65, 78. The initial position of the disk arm is at cylinder 54. The Direction is Left .Implement the SCAN disk scheduling algorithm and calculate the total seek time. Also, plot the sequence of cylinder accesses on a graph.

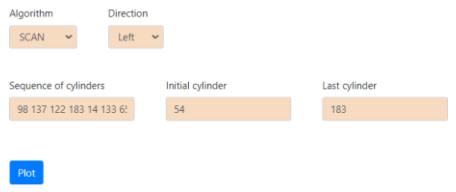

Algorithm: SCAN
Direction: Left
Sequence of cylinders: 98 137 122 183 14 133 6
Initial cylinder: 54
Last cylinder: 183

Fig. 18: SCAN Algorithm

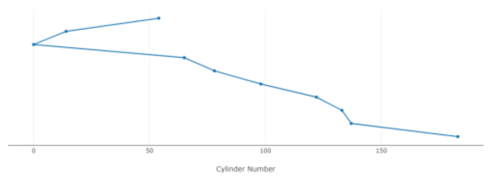


Fig. 19: SCAN Algorithm Graph

Absolute difference between 54 and 14: $|54 - 14| = 40$
Absolute difference between 14 and 0: $|14 - 0| = 14$
Absolute difference between 0 and 65: $|0 - 65| = 65$
Absolute difference between 65 and 78: $|65 - 78| = 13$
Absolute difference between 78 and 98: $|78 - 98| = 20$
Absolute difference between 98 and 122: $|98 - 122| = 24$
Absolute difference between 122 and 133: $|122 - 133| = 11$
Absolute difference between 133 and 137: $|133 - 137| = 4$
Absolute difference between 137 and 183: $|137 - 183| = 46$
Now, we sum up these seek times:
$40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237$
So, the total seek time is 237.

C-SCAN

Example 12: Consider a disk with 200 cylinders numbered from 0 to 199. The requests for disk I/O are as follows:98, 183,37, 122, 14, 124, 65, 67. The initial position of the disk arm is at cylinder 53. The Direction is Right .Implement the C-SCAN disk scheduling algorithm and calculate the total seek time. Also, plot the sequence of cylinder accesses on a graph.

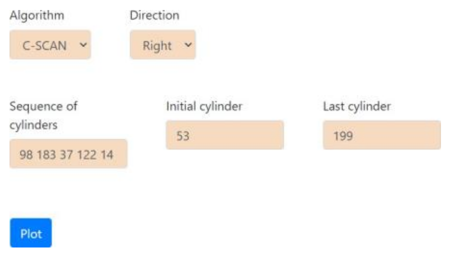

Algorithm: C-SCAN
Direction: Right
Sequence of cylinders: 98 183 37 122 14
Initial cylinder: 53
Last cylinder: 199

Fig. 20: C-SCAN Algorithm

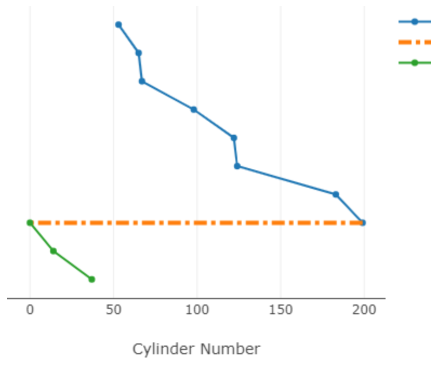


Fig. 21: C-SCAN Algorithm Graph

Absolute difference between 53 and 65: |65 − 53| = 12

Absolute difference between 67 and 65: |67 − 65| = 2

Absolute difference between 98 and 67: |98 − 67| = 31

Absolute difference between 122 and 98: |122 − 98| = 24

Absolute difference between 124 and 122: |124 − 122| = 2

Absolute difference between 183 and 124: |183 − 124| = 59

Absolute difference between 199 and 183: |199 − 183| = 16

Absolute difference between 14 and 0: |0 − 14| = 14

Absolute difference between 37 and 14: |37 − 14| = 23

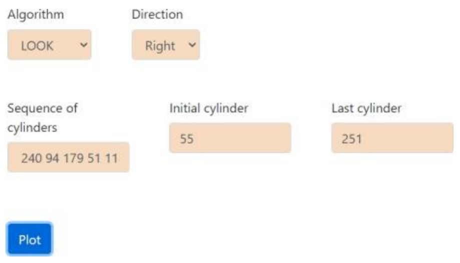
Now, we sum up these seek times:

12 + 2 + 31 + 24 + 2 + 59 + 16 + 14 + 23 = 183

So, the total seek time is 183.

LOOK

Example 13: Let's consider an example of a disk with 251 tracks and a queue of disk access requests in the following order: 240, 94, 179, 51, 118, 15, 137,29 75. The current position of the Read/Write head (C) is at track 55 and moving in the Right direction.Implement the LOOK disk scheduling algorithm and calculate the total seek time. Also, plot the sequence of cylinder accesses on a graph.



Algorithm: LOOK
Direction: Right
Sequence of cylinders: 240 94 179 51 11
Initial cylinder: 55
Last cylinder: 251
Plot

Fig. 22: LOOK Algorithm

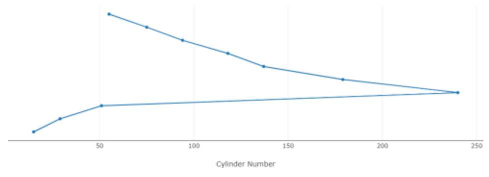


Fig. 23: LOOK Algorithm Graph

Absolute difference between 75 and 54: |75 − 55| = 20

Absolute difference between 118 and 75: |118 − 75| = 43

Absolute difference between 137 and 118: |137 − 118| = 19

Absolute difference between 179 and 137: |179 − 137| = 42

Absolute difference between 240 and 179: |240 − 179| = 61

Absolute difference between 51 and 240: |51 − 240| = 189

Absolute difference between 29 and 51: |29 − 51| = 22

Absolute difference between 29 and 15: |15 − 29| = 14

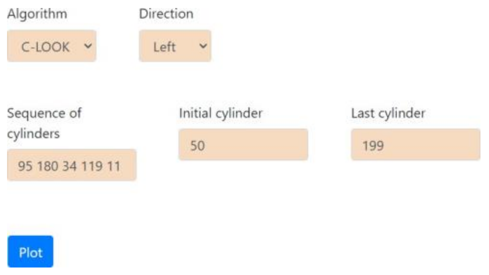
Now, we sum up these seek times:

20 + 43 + 19 + 42 + 61 + 189 + 22 + 14 = 410

So, the total seek time is 410.

C-LOOK

Example 14: Let's consider an example of a disk with 200 tracks and a queue of disk access requests in the following order: 95, 180, 34, 119, 11, 123, 62,64 . The current position of the Read/Write head (C) is at track 50 and moving in the Left direction.Implement the C-LOOK disk scheduling algorithm and calculate the total seek time. Also, plot the sequence of cylinder accesses on a graph.



Algorithm: C-LOOK
Direction: Left
Sequence of cylinders: 95 180 34 119 11
Initial cylinder: 50
Last cylinder: 199
Plot

Fig. 24: C-LOOK Algorithm

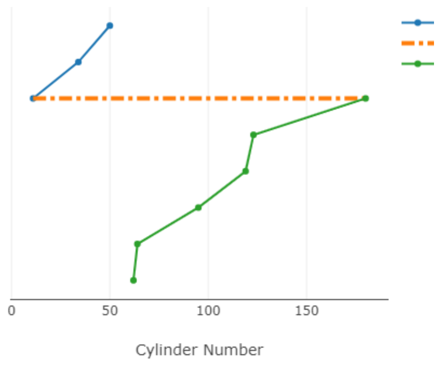


Fig. 25: C-LOOK Algorithm Graph

Absolute difference between 50 and 34: $|50 - 34| = 16$

Absolute difference between 34 and 11: $|34 - 11| = 23$

Absolute difference between 180 and 123: $|180 - 123| = 57$

Absolute difference between 123 and 119: $|123 - 119| = 4$

Absolute difference between 119 and 95: $|119 - 95| = 24$

Absolute difference between 95 and 64: $|95 - 64| = 31$

Absolute difference between 64 and 62: $|64 - 62| = 2$

Now, we sum up these seek times:

$16 + 23 + 57 + 4 + 24 + 31 + 2 = 157$

So, the total seek time is 157.

VI. Discussion

1. Educational Significance: The OS simulator serves as a pivotal educational tool, providing computer science and IT majors with hands-on learning experiences to explore complex OS concepts practically.

2. Virtualization Technology: Leveraging virtualization technology, the simulator enables users to safely test system behavior in virtual environments, mimicking real-world scenarios without risking actual hardware or software components.

3. AI-driven Adaptive Learning: Through AI-driven adaptive learning mechanisms, the simulator tailors the learning experience to individual student needs, enhancing comprehension and retention of OS concepts.

4. Practical Applications: Beyond education, the OS simulator finds practical applications in trouble spot identification, system performance analysis, and improvements to system design and configuration, offering valuable insights for professionals in the field.

VII. Conclusion

An Operating System Simulator provides users a means to interact and disrupt all operating systems components accordingly. It permits them to approach the core ideas and algorithmics applied to OS designing timely. The use of virtualization offers users the possibility to imitate the functionality of a real system thereby, allowing for testing of scenarios, adjustment of setup and observing the real-time effects of adopted settings.

The choice-based adaptive learning technology stands out for its specific application to computer science and information technology (IT) majors and field workers respectively. By providing them with the appropriate platform, they are able to improve cognitive processes such operating systems and how the different parts of the system communicate with each other. Developing simulations and providing analysis of these results allows users to gain useful knowledge on algorithms response speed, discovering trouble spots and identifying areas of improvements.

The simulation of the Operating System is a very useful educational resource aimed at fostering the study and the exploration of the complex operating systems concepts that participants encounter. It regular provides users with room to up their game, to figure out the challenge and mastering how things mechanically work behind the computer systems.

References

[1] Ramamritham, Krithi, and John A. Stankovic. ”Scheduling algorithms and operating systems support for real-time systems.” Proceedings of the IEEE 82.1 (1994): 55-67.

[2] AL-Bakhrani, Ali A., et al. ”Comparative analysis of CPU scheduling algorithms: Simulation and its applications.” International Journal of Advanced Science and Technology 29.3 (2020): 483-494.

[3] Kumar, Anil, et al. ”Operating system security with discrete mathematical structure for secure round-robin scheduling method with intelligent time quantum.” Journal of Discrete Mathematical Sciences and Cryptography 26.5 (2023): 1519-1533.

[4] Abu-Dalbouh, Hussain Mohammad. ”A New Combination Approach to CPU Scheduling based on Priority and Round-Robin Algorithms for Assigning a Priority to a Process and Eliminating Starvation.” International Journal of Advanced Computer Science and Applications 13.4 (2022).

[5] Mostafa, Samih M., and Hirofumi Amano. ”Dynamic round robin CPU scheduling algorithm based on K-means clustering technique.” applied sciences 10.15 (2020): 5134.

[6] Harki, Naji, Abdulraheem Ahmed, and Lailan Haji. ”CPU scheduling techniques: A review on novel approaches strategy and performance assessment.” Journal of Applied Science and Technology Trends 1.1 (2020): 48-55.

[7] Mehmood, Syed Nasir, et al. ”Implementation and experimentation of producer-consumer synchronization problem.” International Journal of Computer Applications 975.8887 (2011): 32-37.

[8] Medhat, Ramy, Borzoo Bonakdarpour, and Sebastian Fischmeister. ”Power-efficient multiple producer-consumer.” 2014 IEEE 28th International Parallel and Distributed Processing Symposium. IEEE, 2014.

[9] Chishti, Mohd Sameen, Chung-Ta King, and Amit Banerjee. ”Exploring half-duplex communication of NFC read/write mode for secure multi-factor authentication.” IEEE Access 9 (2021): 6344-6357.

[10] Kalinovcic, Luka, et al. ”Modified Banker’s algorithm for scheduling in multi-AGV systems.” 2011 IEEE International Conference on Automation Science and Engineering. IEEE, 2011.

[11] Geist, Robert, and Stephen Daniel. ”A continuum of disk scheduling algorithms.” ACM Transactions on Computer Systems (TOCS) 5.1 (1987): 77-92.

[12] Parekh, Harshal Bharatkumar, et al. ”Simulation of a two head disk scheduling algorithm: An algorithm determining the algorithm to be imposed on the heads based on the nature of track requests.” 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, 2017.