Vishwakarma University

Faculty of Science & Technology

Department Of Computer Engineering

Lab Manual

Course Code	Course Name	Teaching Scheme (Hrs. / Week)	Credits
BTECCE22505	Operating Systems	2	1

Course Outcomes:

1. Examine the functions of a contemporary Operating system with respect to convenience, efficiency and the ability to evolve.
2. Demonstrate knowledge in applying system software and tools available in modern operating system, such as threads, system calls, semaphores for software development.
3. Apply various CPU scheduling algorithms and identify deadlock mechanisms to construct solutions to real world problems.
4. Understand the organization of memory and memory management hardware.
5. Analyze I/O and file management techniques for better utilization of secondary memory.

Class : - B-Tech	Branch	: - Computer Engineering
Year : - 2024-25	Prepared By	: - Prof. N. Z. Tarapore

INDEX

Sr. No.	Title of Experiments	Page No
1	Study of Unix commands	3
2	Shell Programming	6
3	Awk Programming	9
4	Reader Writer Problem	11
5	Producer Consumer Problem	14
6	Dining Philosopher Problem	17
7	CPU Scheduling	19
8	Banker's Deadlock Avoidance Algorithm	22
9	Page Replacement Algorithms	25
10	Disk Scheduling	27

Experiment Number: 01

TITLE: Study Of Unix Commands

OBJECTIVES:

1. To understand how to use Unix commands.
2. To understand How and Why they are used in Shell Programming

Problems to be solved in the lab:

1. Change your password to a password you would like to use for the remainder of the semester.
2. Display the system's date.
3. Count the number of lines in the /etc/passwd file.
4. Find out who else is on the system.
5. Direct the output of the man pages for the date command to a file named *mydate*.
6. Create a subdirectory called *mydir*.
7. Move the file *mydate* into the new subdirectory.
8. Go to the subdirectory *mydir* and copy the file *mydate* to a new file called *ourdate*
9. List the contents of *mydir*.
10. Do a long listing on the file *ourdate* and note the permissions.
11. Display the name of the current directory starting from the root.
12. Move the files in the directory *mydir* back to the HOME directory.
13. List all the files in your HOME directory.
14. Display the first 5 lines of *mydate*.
15. Display the last 8 lines of *mydate*.
16. Remove the directory *mydir*.
17. Redirect the output of the long listing of files to a file named *list*.
18. Select any 5 capitals of states in India and enter them in a file named *capitals1*. Choose 5 more capitals and enter them in a file named *capitals2*. Choose 5 more capitals and enter them in a file named *capitals3*. Concatenate all 3 files and redirect the output to a file named *capitals*.
19. Concatenate the file *capitals2* at the end of file *capitals*.

20. Redirect the file *capitals* as an input to the command "wc –l".
21. Give read and write permissions to all users for the file *capitals*.
22. Give read permissions only to the owner of the file *capitals*. Open the file, make some changes and try to save it. What happens ?
23. Create an alias to concatenate the 3 files *capitals1*, *capitals2*, *capitals3* and redirect the output to a file named *capitals*. Activate the alias and make it run.
24. What are the environment variables PATH, HOME and TERM set to on your terminal ?
25. Find out the number of times the string "the" appears in the file *mydate*.
26. Find out the line numbers on which the string "date" exists in *mydate*.
27. Print all lines of *mydate* except those that have the letter "i" in them.
28. Create the file *monotonic* as follows:

    ^a?b?b?c?……..x?y?z$

    Run the egrep command for *monotonic* against /usr/dict/words and search for all 4 letter words.
29. List 5 states in north east India in a file *mystates*. List their corresponding capitals in a file *mycapitals*. Use the *paste* command to join the 2 files.
30. Use the *cut* command to print the 1st and 3rd columns of the /etc/passwd file for all students in this class.
31. Count the number of people logged in and also trap the users in a file using the *tee* command.

APPLICATIONS

1. To enable the user to communicate with the kernel through the command interpreter.
2. Useful in Shell Programming

FAQS

1. What is a pipe?
2. What is a filter?

3. What is the purpose of the grep command?
4. How does input output redirection take place?
5. What is an alias?

Experiment Number: 02

TITLE: Shell Programming

OBJECTIVES:

1. To understand how to perform Shell programming in Unix/Linux.
2. To explain the purpose of shell programs
3. Design and write shell programs of moderate complexity using variables, special variables, flow control mechanisms, operators, arithmetic and functions.

ALGORITHMS

For Palindrome checking:

1. Start.
2. Accept the string from user.
3. Find the actual length of string as len.
4. Initialize a pointer to character in string to 1 and also flag to true.
5. Take the character pointed to by the pointer and the character pointed to by len
6. If the characters are not found, make the flag as false and go to step 9.
7. Decrement variable len by 1 and increment pointer by 1.
8. If the value of len is less than or equal to 1 then repeat from step 5.
9. If the flag is true, display the message that 'String is a palindrome' else display 'String is not palindrome.'
10. Stop.

Test Condition:

- String should not be NULL.

For Bubble sort:

1. Start.
2. Accept how many numbers to be sort say n .
3. Accept the numbers in array say num [].
4. Initialize i and j to 0.
5. While i<n .
6. do

```
        assign j=0
        while j<n
         do
           j=i
           if num[j]<num[i] then
                swap num[i] and num[j]
           end if
        j=j+1
        done
        i=i+1
    done
```

7. Display the sorted list.
8. Stop.

Test Conditions:

- A certain maximum number of elements can be entered.
- Negative numbers are allowed.

For Substring checking:

1. Start.
2. Accept the two strings.
3. Compare two strings character by character.
4. If match is found then store the pointer of first string in an array.
5. Bring counter for second string to the first position.
6. Repeat the steps 3 to 5 until first string gets over.
7. Display the position array if substring exists else display substring does not exist.
8. Stop.

Test condition:

- First string should not be NULL.

APPLICATIONS

1. Useful for integrating our existing applications.
2. Useful for System Administrator.

FAQS

1. What are different control structures used in shell programming?
2. What do you mean by positional parameters & enlist them?
3. What do you mean by shell?

Experiment Number: 03

TITLE: AWK Programming

OBJECTIVES:

To understand how to design & develop AWK Programs in Unix/Linux.

THEORY: The AWK command makes entry into the UNIX system in 1977 to augment the total kit with a suitable report formatting capability. It is named after its authors Aho, Weinberg & Kernighan.

AWK is a pattern scanning and processing language. It is well suited for small projects involving text processing or formatting and has some useful features for small database applications. Its features for pattern description are derived from those of the *grep* family of standard Unix tools.

Syntax:

awk option 'address {action}' file(s)

The action and address together constitutes an 'awk' program. These programs can be one line long or several lines long depending on the application.

A typical and complete 'awk' command specifies an address and an action.

Example

$ awk '/director/ {print }' emp.lst

Output:

neha|director|production

pranav|director|sales

ALGORITHM

1. Start.
2. Input the records in the record file.
3. Start from the first record in the file.
4. Calculate the total marks of the student by adding the values of last 3 columns6
5. Calculate the percentage with total marks.
6. If the percentage is les than 40, give result as fails. If it is between 60 to 65 give result as First Class, Above 66 as Distinction else give result as pass.

7. Repeat steps 3 to 5 for all the records in the record file processed.
8. Stop.

APPLICATIONS

1. It is useful for report generation.

FAQS

1. For what purpose is AWK programming used?
2. Why is AWK more useful compared to shell programming when dealing with database files?
3. What do you mean by $1, $2,$3..etc in AWK programming?
4. What are the different built in variables?

Experiment Number: 04

TITLE: Reader Writer Problem

OBJECTIVES:

1. To understand the solution to the problems of mutual exclusion.
2. To grasp techniques and to develop the skills in the use of the tools: semaphores, threads, mutex in concurrent programming.

THEORY: The problem consists of readers and writers that share a data resource. The readers only want to read from the resource, the writers want to write to it. Obviously, there is no problem if two or more readers access the resource simultaneously. However, if a writer and a reader or two writers access the resource simultaneously, the result becomes indeterminable. Therefore the writers must have exclusive access to the resource.

A data object is to be shared among several concurrent processes. Some of these processes may only want to read content of the shared object, whereas others may want to update the shared object.

We distinguish between these two types of processes by referring to those processes that are interested in only reading as readers and to rest as writers. Obviously, if two readers access the shared data object simultaneously, no adverse effect will result.

1. The **first** readers-writers problem: Requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object.
2. The **second** readers – writers problem: Requires that, once a writer is ready, that writer performs its write as soon as possible. At given time, there is only one writer and any number of readers. When a writer is writing, the readers cannot enter into the database .The readers need to wait until the writer finishes to write to the database. Once a reader succeeds in reading the database, subsequent readers can enter into the critical section without waiting for the precedent reader finish to read. On the other hand, a writer who arrives later than the reader who is reading currently is required to wait till the last reader finishes read. Only when

the last reader finishes reading, can the writer enter into the critical section and is able to write to the database.

SEMAPHORE:

- It can be used to restrict access to the database under certain conditions.
- Semaphore allows a sleep and wakeup mutual exclusion policy to be implemented.
- Basically, a semaphore is a new type of variable. A semaphore can have a value of 0(meaning no wakeups are saved) or a positive integer value, indicating the number of sleeping processes.
- Two different operations can be performed on a semaphore, down and up, corresponding to Sleep and Wakeup.

MUTEX:

A mutex is mutual exclusion lock. Only one thread can hold the lock. Mutexes are used to protect data or other resources from concurrent access. A mutex has attributes, which specify the characteristics of the mutex.

ALGORITHMS

Using mutex and threads:

1. Start.
2. Create two threads associated with processes update and display.
3. Declare mutex variable timer_lock and initialize to 0 using pthread_mutex_init function.
4. Join two threads. Initiate processes update and display.

Update ( )

1. Lock access to critical sections using mutex timer_lock.
2. Update the values of variables seconds, minutes, and hours.
3. Unlock the access to critical sections using mutex.

Display( )

1. Lock the access to critical sections using mutex timer_lock.
2. Display the values of variables hours, minutes, seconds.

3. Unlock the access to critical sections using mutex timer_lock.

Using semaphore and thread :

1. Start.
2. Create two threads associated with processes writer and reader.
3. Initialize two semaphore variables pread to 0 and pwrite to 1.
4. Join two threads. Initiate processes writer and reader.
5. Destroy semaphores pread, pwrite using sem_destroy function.
6. Stop.

Reader ( )

1. Check whether the value of semaphore pread is 1.
2. If yes, decrement the value of the semaphore to 0.
3. Read the value of seconds, minutes, hours.
4. Display the values of variables hours, minutes, seconds.
5. Increment the value of the semaphore pwrite to 1.
6. Stop.

Writer ( )

1. Check whether the value of semaphore pwrite is 1.
2. If yes, decrement the value of the semaphore to 0.
3. Update the value of seconds, minutes, hours.
4. Increment the value of the semaphore pread to 1.
5. Stop.

APPLICATIONS

1. Applicable in all applications where synchronization is used such as all Operating systems.

FAQS

1. What is difference between thread and process?
2. What mean by is semaphore?
3. What is mean by critical section?
4. How will you change the solution for n readers and n writer?

Experiment Number: 05

TITLE: Producer Consumer Problem

OBJECTIVES:

1. To understand problems of mutual exclusion and Producer Consumer Problem.
2. To grasp techniques and to develop the skills in the use of the tools: semaphores, threads, and mutex in concurrent programming.

THEORY: The producer-consumer problem illustrates the need for synchronization in systems where many processes share a resource. In this problem, two processes share a fixed size buffer. One process produces information and puts it in the buffer, while the other process consumes information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently.

ALGORITHMS

Using Mutex and Thread

1. Start.
2. Declare two thread variables of pthread_t structure.
3. Create two threads associated with producer and consumer processes using pthread_create function.
4. Declare one mutex variable of pthread_mutex_t structure mutex.
5. Initialize the mutex variable using pthread_mutex_init function.
6. Join two threads using pthread_join function.
7. Initiate two processes.
8. Stop.

Producer ( )

1. Producer locks the mutex variable mutex using pthread_mutex_lock function.
2. Producer executes the critical section.

3. Then after producing the string, the producer unlocks the mutex using the pthread_mutex_unlock function and makes it available for the consumer.

Consumer ( )

1. Consumer will wait for mutex variable.
2. Consumer locks the mutex variable mutex using pthread_mutex_lock function.
3. Consumer executes the critical section.
4. Then after consuming the string, consumer unlocks the mutex using pthread_mutex_unlock function and makes it available for the producer.

Using semaphore and thread:

1. Declare two thread variables of pthread_t structure.
2. Create two threads associated with producer and consumer processes using pthread_create function.
3. Declare two semaphore variables of sem_t structure bfull and bempty.
4. Initialize the semaphore variable empty to 0 and full to 1 using sem_init function.
5. Join two threads using pthread_join function.
6. Initiate two processes.
7. Destroy both the semaphore variables using sem_destroy function.
8. Stop.

Producer ( )

1. If semaphore variable bfull is 1, then the producer will wait on empty using the sem_wait function.
2. Else the producer will produce the string and execute the critical section. Then signal the bfull semaphore variable using the sem_post function on full and unblock the consumer thread.

Consumer ( )

1. If there is no data to consume, then the thread waits on the semaphore variable bfull using the sem_wait function.

2. Else the consumer executes the critical section and consumes the data. Then the consumer signals the bempty variable using sem_post function and unblocks the producer thread.

Test Conditions: Maximum string length should not be exceeded**.**

APPLICATIONS

1. Applicable in all applications where synchronization is used such as all Operating systems.

FAQS

1. How will you change the solution for *n* producers and *n* consumers?
2. What do you mean by mutual Exclusion condition?

Experiment Number: 06

TITLE: Dining Philosopher Problem

OBJECTIVES:

How to solve the synchronization problem using multithreading

THEORY: The Dining Philosopher problem is stated as follows

Five philosophers are seated around a table. Each philosopher has a plate of spaghetti. A philosopher must eat the spaghetti with two chopsticks. Between every two plates there is a chopstick. The life of a philosopher consists of alternate periods of eating and thinking. When a philosopher gets hungry, he tries to acquire his left and right chopsticks, one at time, in either order. If successful in acquiring two chopsticks, he eats for a while, then puts down the chopsticks and continues to think. The key question is: can you write a program for each philosopher that does what it is supposed to do and never gets stuck?

The dining philosophers problem is useful for modeling processes that are competing for exclusive access to a limited number of resources, such as tape drives or other I/O devices.

ALGORITHMS

Using Threads and Semaphores :

1. Declare an array of semaphores. Each element in array is associated with each philosopher.
2. Declare one thread for each of the five philosopher processes.
3. Associate every thread with each philosopher procedure.

Procedure Philosopher:

1. Decrement the value of the associated semaphore with the philosopher.
2. Check the state of philosopher. If the state is thinking, his state is changed to hungry.

3. If state is hungry, the states of left and right philosophers are checked. If both of them are not eating, the philosopher can eat. Else, the philosopher is still hungry.
4. If philosopher is eating, his state is again changed to thinking.
5. The value of semaphore is increased by one.

Procedure philosopher:

1. Check the present the state of the respective philosopher.
2. If the state is thinking, change the state to hungry.
3. If the state is hungry, check the state of left and right philosophers
4. If both philosophers are not eating, change the state of philosopher to eating.
5. Else, the philosopher is hungry.
6. If the philosopher is eating, change state to thinking.

Test conditions:

- Philosophers placed alternately can eat simultaneously.
- Each philosopher is allowed to eat for a maximum of 3 seconds, so as to avoid starvation.

APPLICATIONS

1. Applicable in all applications where synchronization is used such as all Operating systems.

FAQS

1. What do you mean by multithreading?
2. What is the Dining Philosopher's problem?
3. If there are *n* philosophers, what is the maximum number of philosophers that can eat at a time?

Experiment Number: 07

TITLE: CPU Scheduling

OBJECTIVES:

To study different process scheduling algorithms.

THEORY:

Scheduling decisions may take place when a process switches from:

1.Running to waiting.

2.Running to ready.

3.Waiting to ready.

4.Running to terminate.

Non-preemptive scheduling:

The current process keeps the CPU until it releases the CPU by either terminating or by switching to a waiting state.

Non-preemptive scheduling occurs only under situations 1 and 4, it does not require special hardware (timer).

Preemptive scheduling:

The currently running process may be interrupted and moved to the ready state by the operating system. It requires special hardware (timer), mechanisms to coordinate access to shared data.

Definitions:

- Throughput Number of processes/time unit.
- Turnaround Time it take to execute a process from start to finish.
- Waiting Time Total time spent in the ready queue.
- Response time Amount of time it takes to start responding (average, variance).

ALGORITHMS

First Come First Serve (FCFS):

1. Accept burst time and arrival time for every process entered by user.
2. Compare the arrival time of all processes.
3. Sort processes in ascending order with respect to their arrival time.
4. Execute all processes in sorted order.
5. Calculate the finish time of each process using the formula.
   FT = start time + burst time.
6. Calculate the turnaround and waiting time for all processes.
7. Display finish time, turnaround time, waiting time, Gantt chart.
8. Stop.

Shortest Job First (SJF):

1. Accept the number of processes from the user.
2. Accept arrival time and burst times for each process.
3. Start with arrival time.
4. For every arrival time, check which jobs are available.
5. Select job with shortest burst time.
6. Complete the selected process.
7. Continue steps 4 & 6 until all processes are complete.
8. Display finish time, turnaround time, waiting time with Gantt chart.
9. Stop.

Round Robin:

1. Accept the number of processes from the user.
2. Accept arrival time and burst times for each process.
3. Start with arrival time
4. Execute all processes present at arrival time for one time slot.
5. Increment arrival time.
6. Continue steps 3 to 4 until all the processes are complete.

7. Display finish time, turnaround time, waiting time, Gantt chart.
8. Stop.

APPLICATIONS

Operating system use scheduling algorithms in order to provide good response to users.

FAQS

1. Why is scheduling used?
2. What is the difference between preemptive & non- preemptive scheduling?
3. Which algorithm is more useful & why?
4. What do you mean by time quantum?

Experiment Number: 08

TITLE: Banker's Deadlock Avoidance Algorithm

OBJECTIVES:

1. To simulate deadlock avoidance.
2. To check if state is safe or not.

THEORY:

Deadlock: In operating systems or databases, a situation in which two or more processes are prevented from continuing while each waits for resources to be freed by the continuation of the other.

Deadlock Characterization: Deadlock can arise if four conditions hold simultaneously. (All four must hold)

1. Mutual exclusion: Only one process at a time can use a resource
2. Hold and Wait: A process holding at least one resource is waiting to acquire additional resources held by other processes.
3. No preemption: A resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. Circular Wait: There exists a set $\{P_0, P_1,\ldots.P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1,P_1 is waiting for a resource that is held by P_2..P_{n-1} is waiting for a resource that is held by P_n ,and P_n is waiting for a resource that is held by P_0.

Deadlock Prevention: Restrain the ways that processes can make resource requests:

Mutual Exclusion- not required for sharable resources; must hold for non-sharable resources

Hold and wait-must guarantee that whenever a process requests a resource, it does not hold any other resources.

- Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.

- Low resource utilization; starvation possible

No Preemption-

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- Preempted resources are added to the list of resources for which the process is waiting.
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Circular wait-Impose a total ordering of all resources types and require that each process request resources in a increasing order of enumeration.

Deadlock Avoidance:

Requires that the system has additional information in advance.

- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need
- The deadlock –avoidance algorithm dynamically examines the resource allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

ALGORITHMS

1. Start.
2. Calculate the current need for each process, for each resource from the data entered by user.
3. For a process, check if current available resources satisfy all current needs.
4. If all are satisfied completes the process and adds all its current allocations to available resources.
5. If all are not satisfied, check for next process.
6. Repeat from step 3, for as many times as there are processes.
7. If all processes have been completed, system is in safe state and display safe sequence. Else system is not in safe state.

8. Stop.

Test Condition: Safe sequence is not unique. A different safe sequence may also be possible.

APPLICATIONS

Can be used in a system where prior information regarding usage of resources for different processes is known in advance.

FAQS

1. What do you mean by deadlock?
2. What are 4 conditions necessary for deadlock existence?
3. What are time complexities of deadlock avoidance and deadlock detection algorithm?

Experiment Number: 09

TITLE: Page Replacement Algorithms

OBJECTIVES:

To study different Page Replacement algorithms

THEORY:

Page: One of numerous equally sized chunks of memory.

Page Replacement: This policy determines which page should be removed from main memory so that page from secondary memory replaces it.

ALGORITHMS

First In First Out (FIFO):

1. Start.
2. Accept the sequence of requirement of pages.
3. Initialize a pointer to the first page.
4. Check if the page is already present in the main memory.
5. If present, repeat from step 4, for next page.
6. If the page is not present, remove the page, which has entered the main memory first and put this page.
7. Increment the pointer to page to be replaced to next page.
8. Repeat from step 4 till all pages in sequence are over.
9. Stop.

Least Recently Used (LRU):

1. Start.
2. Accept the sequence of requirement of pages.
3. Initialize a count for each page in main memory to zero.
4. Check if the page is already present in the main memory.
5. If the page is not present, find the page in memory with maximum count, which is the least recently used and replace it.
6. Set it's count to zero and increment others count.

7. If found, set its count to zero and increment others count.
8. Repeat from step 4 till all pages in sequence are over.
9. Stop.

Optimal:

1. Start.
2. Accept the sequence of requirement of pages.
3. Initialize a count for each page in main memory to zero.
4. Check if the page is already present in the main memory.
5. If the page is not present, find the page which will not be used for longest period.
6. Repeat steps 4 till all pages in sequence are over.
7. Stop.

APPLICATIONS

1. Used in memory management in Operating Systems.

FAQS

1. What do you mean by virtual memory?
2. What is segmentation?
3. What is a translation look aside buffer?
4. What is thrashing?
5. What is Belady's anomaly?

Experiment Number: 10

TITLE: Disk Scheduling Algorithms

OBJECTIVES:

To study different disk scheduling algorithms

THEORY:

In multiprogramming systems several different processes may want to use the system's resources simultaneously. For example, processes will contend to access an auxiliary storage device such as a disk. The disk drive needs some mechanism to resolve this contention, sharing the resource between the processes fairly and efficiently.

First Come First Serve (FCFS)

The disk controller processes the I/O requests in the order in which they arrive, thus moving backwards and forwards across the surface of the disk to get to the next requested location each time. Since no reordering of request takes place the head may move almost randomly across the surface of the disk. This policy aims to minimize response time with little regard for throughput.

Shortest Seek Time First (SSTF)

Each time an I/O request has been completed the disk controller selects the waiting request whose sector location is closest to the current position of the head. The movement across the surface of the disk is still apparently random but the time spent in movement is minimized. This policy will have better throughput than FCFS but a request may be delayed for a long period if many closely located requests arrive just after it.

SCAN

The drive head sweeps across the entire surface of the disk, visiting the outermost cylinders before changing direction and sweeping back to the innermost cylinders. It selects the next waiting requests whose location it will reach on its path backwards and forwards across the disk. Thus, the movement time should be less than FCFS but the policy is clearly fairer than SSTF.

Circular SCAN (C-SCAN)

C-SCAN is similar to SCAN but I/O requests are only satisfied when the drive head is traveling in one direction across the surface of the disk. The head sweeps from the innermost cylinder to the outermost cylinder satisfying the waiting requests in order of their locations. When it reaches the outermost cylinder it sweeps back to the innermost cylinder without satisfying any requests and then starts again

ALGORITHMS

First Come First Serve (FCFS):

1. Start.
2. Accept the number of tracks *n*.
3. Accept the requested tracks and store it in the track [ ].
4. Consider the first value of track[ ] as starting track.
5. Process the track values in given order to calculate difference as
   trackdiff [i] = track[i] - track[i+1] .
6. Calculate totaldiff = totaldiff + trackdiff[i].
7. Calculate average seek time.
8. Display the value of average seek time.
9. Stop.

Shortest Service Time First (SSTF):

1. Start.
2. Accept the number of tracks *n.*
3. Accept the requested tracks and store it in track[ ].
4. Consider the first value of track[ ] as starting track .
5. First process all the tracks which are having value less than starting track in decreasing order of tracks as trackdiff[i] = track[i]-track[i+1] .
6. Then process all the tracks which are having value greater than starting in increasing order of track as trackdiff[i] = track[i] - track[i+1].
7. Calculate totaldiff = totaldiff + trackdiff[i]
8. Calculate average seek time.
9. Display the value of average seeks time.
10. Stop.

SCAN:

1. Start.
2. Accept the number of tracks *n*.
3. Accept the requested tracks and store it in track[ ]
4. Consider the first value of track[ ] as starting track .
5. First process all the tracks which are having value greater than starting track in increasing order of track as trackdiff[i] = track[i] - track[i+1]
6. then process all the tracks which are having value less than starting track in decreasing order of track as trackdiff[i] = track[i] - track[i+1]
7. Calculate totaldiff = totaldiff + trackdiff[i]
8. Calculate average seek time.
9. Display the value of average seek time.
10. Stop

C-SCAN:

1. Start.
2. Accept the number of tracks *n.*
3. Accept the requested tracks and store it in track[ ]
4. Consider the first value of track[ ] as starting track .
5. First process all the tracks which are having value greater than starting track in increasing order of track as trackdiff[i]= track[i] - track[i+1]
6. Then process all the tracks which are having value less than starting track in increasing order of track(Wrap around to starting track) as trackdiff[i] = track[i] - track[i+1]
7. Calculate totaldiff = totaldiff + trackdiff[i]
8. Calculate average seek time.
9. Display the value of average seek time.
10. Stop

APPLICATIONS

1. Applicable in Multimedia application to reduce disk access.

FAQS

1. What do you mean by seek time?
2. What is difference between SCAN & C-SCAN method?
3. Give an analysis of comparison of 4 algorithms.