Department of Computer Science & Engineering

OPERATING SYSTEM LAB

(CSC211)

NLHC - LAB 1

OPERATING SYSTEM LAB

(CSC211)

LAB MANUAL

IV Semester B.Tech (CSE)

Winter Semester (2021-2022)

Department of Computer Science and Engineering
Indian Institute of Technology (Indian School of Mines)

List of Experiments

Lab 1

Basics of UNIX commands and Implementation of Shell Programming

Linux Command	*DOS Command*	*Description*
pwd	cd	"Print Working Directory". Shows the current location in the directory tree.
cd	cd, chdir	"Change Directory". When typed all by itself, it returns you to your home directory.
cd directory	cd directory	Change into the specified directory name. Example: cd /usr/src/linux
cd ~		"~" is an alias for your home directory. It can be used as a shortcut to your "home", or other directories relative to your home.
cd ..	cd..	Move up one directory. For example, if you are in /home/vic and you type "cd ..", you will end up in /home.
cd -		Return to previous directory. An easy way to get back to your previous location!
ls	dir /w	List all files in the current directory, in column format.
ls directory	dir directory	List the files in the specified directory. Example: ls /var/log
ls -l	dir	List files in "long" format, one file per line. This also shows you additional info about the file, such as ownership, permissions, date, and size.
ls -a	dir /a	List all files, including "hidden" files. Hidden files are those files that begin with a ".", e.g. The .bash_history file in your home directory.
ls -ld directory		A "long" list of "directory", but instead of showing the directory contents, show the directory's detailed information. For example, compare the output of the following two commands: ls -l /usr/bin ls -ld /usr/bin
ls /usr/bin/d*	dir d*.*	List all files whose names begin with the letter "d" in the /usr/bin directory.

Q1. Write a shell program to add two numbers.

ALGORITHM:

1. Read first number
2. Read second number
3. Add both numbers in third variable.
4. Print the third variable

Q2. write a shell program to find a number is even or odd

ALGORITHM:

1. Read the number
2. Use the condition of checking if the number is even by using second variable
3. Check if the variable equals 0
4. Print the result if the number is even or not

Q -3write a shell program for fibbonacci series.

ALGORITHM:

1. Start
2. Declare variable i, a, b, show.
3. Initialize a=0, b=1 and show=0
4. Enter the number of terms of series for printing.
5. Print first two terms of series.
6. Use loop for: show=a+b. a=b. b=show. Increment i.
7. End.

Questions:

1. Execute various UNIX commands.
2. Write a shell program for Fibonacci series.
3. What is the use of cat commands?
4. Define Operating Systems.
5. What is the use of filter/grep/pipe commands?
6. How is unix different from windows?
7. What is unix?
8. What is the file structure of unix?
9. What is a kernel?

10. What is the difference between multi-user and multi-tasking?
11. Differentiate relative path from absolute path.
12. What are the differences among a system call, a library function, and a UNIX command?

Lab 2

Implementation of Process and thread (Life cycle operations)

Process

Fork system call is used for creating a new process, which is called child process.

The child process runs concurrently with the process that makes the **fork()** call (parent process).

After a new child process is created, both processes will execute the next instruction following the **fork()** system call. A child process uses the same PC (Program Counter), same CPU registers, same open files which use in the parent process.

It takes no parameters and returns an integer values returned by **fork()** as given below.

Negative Value*:* creation of a child process was unsuccessful.

Zero*:* Returned to the newly created child process.

Positive value*:* Returned to parent or caller. The value contains process ID of newly created child process.

Thread

pthread_create: This is used to create a new thread.

Syntax:

```
int pthread_create(pthread_t * thread, const pthread_attr_t * attr, void * (*start_routine)(void *),
                   void *arg);
```

Parameters:

- **thread:** pointer to an unsigned integer value that returns the thread id of the thread created.
- **attr:** pointer to a structure that is used to define thread attributes like detached state, scheduling policy, stack address, etc. Set to NULL for default thread attributes.
- **start_routine:** pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *. The function has a single attribute but if multiple values need to be passed to the function, a struct must be used.
- **arg:** pointer to void that contains the arguments to the function defined in the earlier argument

pthread_exit: This is used to terminate a thread

Syntax:

```
void pthread_exit(void *retval);
```

Parameters: This method accepts a mandatory parameter **retval** which is the pointer to an integer that stores the return status of the thread terminated. The scope of this variable must be global so that any thread waiting to join this thread may read the return status.

pthread_join: used to wait for the termination of a thread.

Syntax:

```
int pthread_join(pthread_t th, void **thread_return);
```

Parameter: This method accepts following parameters:

- **th:** thread id of the thread for which the current thread waits.
- **thread_return:** pointer to the location where the exit status of the thread mentioned in th is stored.

pthread_self: This is used to get the thread id of the current thread.

Syntax:

```
pthread_t pthread_self(void);
```

pthread_equal: It compares whether two threads are the same or not. If the two threads are equal, the function returns a non-zero value otherwise zero.

Syntax:

```
int pthread_equal(pthread_t t1, pthread_t t2);
```

Parameters: This method accepts following parameters:

- t1: the thread id of the first thread
- t2: the thread id of the second thread

pthread_cancel: used to send a cancellation request to a thread
Syntax:

```
int pthread_cancel(pthread_t thread);
```

Parameter: This method accepts a mandatory parameter **thread** which is the thread id of the thread to which cancel request is sent.

pthread_detach: used to detach a thread. A detached thread does not require a thread to join on terminating. The resources of the thread are automatically released after terminating if the thread is detached.

Syntax:

```
int pthread_detach(pthread_t thread);
```

Parameter: This method accepts a mandatory parameter **thread** which is the thread id of the thread that must be detached.

Assignment Questions:

1. Write a C/C++ program for process creation and termination.
2. Write a C/C++ program for thread creation and termination.

Lab 3

Implementation of CPU Scheduling: i) FCFS, ii) Shortest Job First iii) Shortest Remaining Time First and iv) Priority based

i. First Come First Serve (FCFS)

1. Start the program.
2. Get the number of processes and their burst time.
3. Initialize the waiting time for process 1 and 0.
4. Process for(i=2;i<=n;i++), wt.p[i]=p[i-1]+bt.p[i-1].
5. The waiting time of all the processes is summed then average value time is calculated.
6. The waiting time of each process and average times are displayed
7. Stop the program

ii. Shortest Job First (SJF)

1. Start the program. Get the number of processes and their burst time.
2. Initialize the waiting time for process 1 as 0.
3. The processes are stored according to their burst time.
4. The waiting time of all the processes summed and then the average time is calculated
5. The waiting time of each process and average time are displayed.
6. Stop the program.

iii. Shortest Remaining time first

1. Get the number of process and their burst time.
2. Traverse until all process gets completely executed.
    i. Find process with minimum remaining time at every single time lap.
    ii. Reduce its time by 1.
    iii. Check if its remaining time becomes 0
    iv. Increment the counter of process completion.
    v. Completion time of current process = current_time +1;
    vi. Calculate waiting time for each completed process.
    vii. wt[i]= Completion time - arrival_time-burst_time
    viii. Increment time lap by one.
3. The waiting time for each process are displayed.
4. Stop the program.

iv. Priority based

1. Start the program.
2. Read burst time, waiting time, turn the around time and priority.
3. Initialize the waiting time for process 1 and 0.
4. Based up on the priority process are arranged
5. The waiting time of all the processes is summed and then the average waiting time
6. The waiting time of each process and average waiting time are displayed based on the priority.
7. Stop the program

Assignment Questions:

1. Write a C/C++ program to simulate the following non preemptive CPU scheduling algorithms to find average turnaround time and average waiting time. (Provide user inputs)

a) FCFS

b) SJF

c) SRTF

d) Priority Based

Sample Input:

Process	Arrival Time	Burst Time	Priority
P0	0	4	2
P1	1	3	3
P2	2	1	4
P3	3	5	5
P4	4	2	5

Viva Questions:

1. Why CPU scheduling is required?
2. CPU performance is measured through _____.
3. What is a criterion to evaluate a scheduling algorithm?

Lab 4

Implementation of CPU Scheduling (Pre-emptive Processes): i) Round Robin ii) Longest job first and iii) Longest Remaining Time First

i. Round Robin

1. Get the number of process and their burst time.
2. Initialize the array for Round Robin circular queue as '0'.
3. The burst time of each process is divided and the quotients are stored on the round Robin array.
4. According to the array value the waiting time for each process and theaverage time are calculated as line the other scheduling.
5. The waiting time for each process and average times are displayed.
6. Stop the program.

ii. Longest Job First

This algorithm is a non-preemptive scheduling algorithm. This algorithm is based upon the burst time of the processes. The processes are put into the ready queue based on their burst times i.e., in descending order of the burst times.

Algorithm:

- **Step-1:** First, sort the processes in increasing order of their Arrival Time.
- **Step-2:** Choose the process having highest Burst Time among all the processes that have arrived till that time. Then process it for its burst time. Check if any other process arrives until this process completes execution.
- **Step-3:** Repeat the above both steps until all the processes are executed.

Disadvantages-

- This algorithm gives very high average waiting time and average turn-around time for a given set of processes.
- It may happen that a short process may never get executed and the system keeps on executing the longer processes.
- It reduces the processing speed and thus reduces the efficiency and utilization of the system.

iii. LRTF

Algorithm

- **Step-1:** Create a structure of process containing all necessary fields like AT (Arrival Time), BT(Burst Time), CT(Completion Time), TAT(Turn Around Time), WT(Waiting Time).
- **Step-2:** Sort according to the AT;
- **Step-3:** Find the process having Largest Burst Time and execute for each single unit. Increase the total time by 1 and reduce the Burst Time of that process with 1.
- **Step-4:** When any process have 0 BT left, then update the CT(Completion Time of that process CT will be Total Time at that time).
- **Step-2:** After calculating the CT for each process, find TAT and WT.

Assignment Questions:

1. Write a C/C++ program to simulate the following preemptive CPU scheduling algorithms to find average turnaround time and average waiting time. (Take User input)

Time Quantum = 2

a) Round Robin

b) LJF

c) LRTF

Sample Input:

Process	Arrival Time	Burst Time
P0	0	4
P1	1	7
P2	2	1
P3	3	8
P4	4	5

Viva Questions:

1. In preemptive scheduling, a running process is _____ if another high priority process enters the ready queue.
2. Super computers typically employ _____.

Lab 5

Implementation of CPU Scheduling: Highest Response Ratio Next (HRRN), (ii) Multilevel Queue (iii) Multilevel Feedback Queue

(i) Highest Response Ratio Next (HRRN)

In HRRN Scheduling,

- Out of all the available processes, CPU is assigned to the process having highest response ratio.
- In case of a tie, it is broken by FCFS Scheduling.
- It operates only in non-preemptive mode.

Calculating Response Ratio-

Response Ratio (RR) for any process is calculated by using the formula-

$$\text{Response Ratio} = \frac{W + B}{B}$$

Where-
W = Waiting time of the process so far
B = Burst time or Service time of the process

HRRN Scheduling Example

Process Id	Arrival time	Burst time
P0	0	3
P1	2	6
P2	4	4
P3	6	5
P4	8	2

Gantt Chart

Solution:

Step 1:

- At t = 0, only the process P0 is available in the ready queue.
- So, process P0 executes till its completion.

Step 2:

- At t = 3, only the process P1 is available in the ready queue.
- So, process P1 executes till its completion.

Step 3:

- At t = 9, the processes P2, P3 and P4 are available in the ready queue.
- The process having the highest response ratio will be executed next.

The response ratio are-

- Response Ratio for process P2 = [(9-4) + 4] / 4 = 9 / 4 = 2.25
- Response Ratio for process P3 = [(9-6) + 5] / 5 = 8 / 5 = 1.6
- Response Ratio for process P4 = [(9-8) + 2] / 2 = 3 / 2 = 1.5

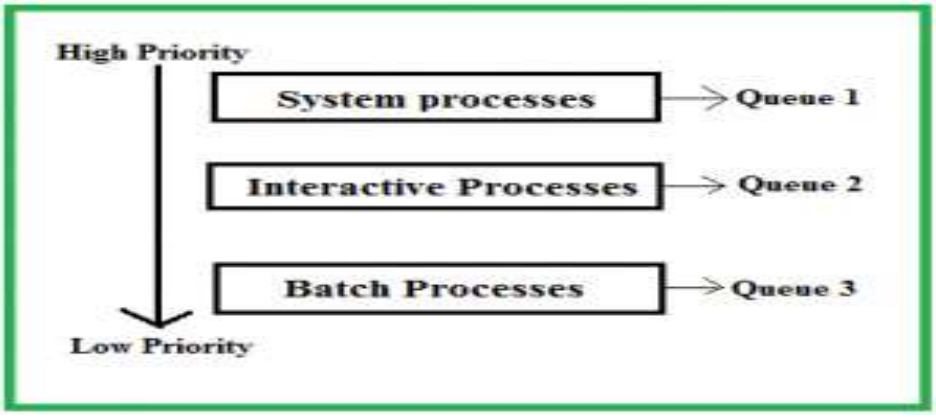
Since process P2 has the highest response ratio, so process P2 executes till completion.

Step 4: Follow the same rule for next process available.

Algorithm:

1. Input the number of processes, their arrival times and burst times.
2. Sort them according to their arrival times.
3. At any given time calculate the response ratios and select the appropriate process to be scheduled.
4. Calculate the turn around time as completion time – arrival time.
5. Calculate the waiting time as turn around time – burst time.
6. Turn around time divided by the burst time gives the normalized turn around time.
7. Sum up the waiting and turn around times of all processes and divide by the number of processes to get the average waiting and turn around time.

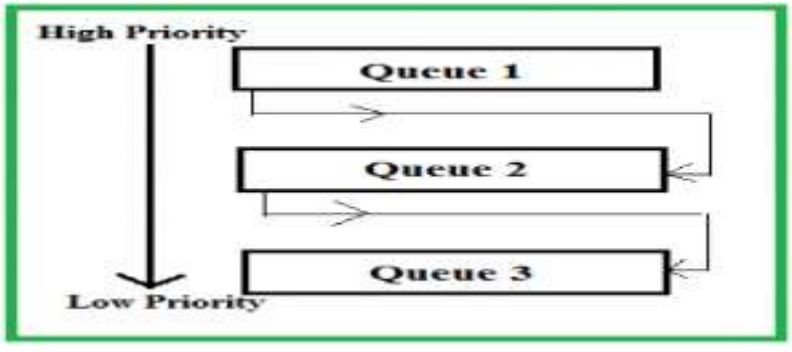
(ii) Multilevel Queue



There are two ways to do so-

1. **Fixed priority preemptive scheduling method –** Each queue has absolute priority over the lower priority queue. Let us consider following priority order **queue 1 > queue 2 > queue 3**. According to this algorithm, no process in the batch queue(queue 3) can run unless queues 1 and 2 are empty. If any batch process (queue 3) is running and any system (queue 1) or Interactive process(queue 2) entered the ready queue the batch process is preempted.
2. **Time slicing** – In this method, each queue gets a certain portion of CPU time and can use it to schedule its own processes. For instance, queue 1 takes 50 percent of CPU time queue 2 takes 30 percent and queue 3 gets 20 percent of CPU time.

(iii). Multilevel Feedback Queue



Let us suppose that queues 1 and 2 follow round robin with time quantum 4 and 8 respectively and queue 3 follow FCFS. One implementation of MFQS is given below –

1. When a process starts executing then it first enters queue 1.
2. In queue 1 process executes for 4 units and if it completes in this 4 unit or it gives CPU for I/O operation in this 4 unit then the priority of this process does not change and if it again comes in the ready queue then it again starts its execution in Queue 1.

3. If a process in queue 1 does not complete in 4 units then its priority gets reduced and it shifted to queue 2.
4. Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 units.In a general case if a process does not complete in a time quantum then it is shifted to the lower priority queue.
5. In the last queue, processes are scheduled in an FCFS manner.
6. A process in a lower priority queue can only execute only when higher priority queues are empty.
7. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.

Assignment Questions:

1. Write a C/C++ program to simulate HRRN CPU scheduling algorithms to find average turnaround time and average waiting time. (Take User input)

2. Write a C/C++ program to simulate Multilevel Feedback Queue CPU scheduling algorithms to find average turnaround time and average waiting time, where queues 1 and 2 follow round robin with time quantum 4 and 8, respectively and queue 3 follow FCFS.
3. Write a C/C++ program to simulate Multilevel Queue CPU scheduling algorithms to find the average turnaround time and average waiting time (Processes P0, P1, P4 in Queue 1 and Processes P2, P3 in Queue 2).

Sample Input:

Process	Arrival Time	Burst Time
P0	1	3
P1	3	6
P2	5	8
P3	7	4
P4	8	5

Viva Questions:

1. What are CPU-bound I/O-bound processes?

Lab 6

PRODUCER-CONSUMER PROBLEM USING SEMAPHORES

Producer-Consumer Problem is also known as bounded buffer problem. The Producer-Consumer Problem is one of the classic problems of synchronization.

There is a buffer of N slots and each slot is capable of storing one unit of data.

There are two processes running, i.e. Producer and Consumer, which are currently operated in the buffer.

There are certain restrictions/conditions for both the producer and consumer process, so that data synchronization can be done without interruption. These are as follows:

- The producer tries to insert data into an empty slot of the buffer.
- The consumer tries to remove data from a filled slot in the buffer.
- The producer must not insert data when the buffer is full.
- The consumer must not remove data when the buffer is empty.
- The producer and consumer should not insert and remove data simultaneously.

For solving the producer-consumer problem, three semaphores are used:

- m(mutex) : A binary semaphore which is used to acquire and release the lock.
- empty(), a counting semaphore whose initial value is the number of slots in the buffer, since initially, all slots are empty.
- full, a counting semaphore, whose initial value is 0.

Implementation of producer code

```
void Producer(){
   do{
      //wait until empty > 0
      wait(Empty);
      wait(mutex);
      add()
      signal(mutex);
      signal(Full);
   }while(TRUE);
```

```
}
```

Implementation of consumer code

```
void Producer(){
   do{
      //wait until empty > 0
      wait(full);
      wait(mutex);
      consume()
      signal(mutex);
      signal(empty);
  }while(TRUE);
}
```

ALGORITHM:

1. Declare variable for producer & consumer as pthread-t-tid produce tid consume.
2. Declare a structure to add items, semaphore variable set as struct.
3. Read number the items to be produced and consumed.
4. Declare and define semaphore function for creation and destroy.
5. Define producer function.
6. Define consumer function.
7. Call producer and consumer.
8. Stop the execution.

READER WRITER PROBLEM

The readers-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object.

The readers-writers problem is used to manage synchronization so that there are no problems with the object data. For example - 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.

To solve this situation, a writer should get exclusive access to an object i.e. when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the object at the same time.

This can be implemented using semaphores. The codes for the reader and writer process in the reader-writer problem are given as follows –

Reader Process

The code that defines the reader process is given below −

```
wait (mutex);
  rc ++;
  if (rc == 1)
  wait (wrt);
  signal(mutex);

   READ THE OBJECT
  wait(mutex);
  rc --;
  if (rc == 0)
  signal (wrt);
      signal(mutex);
```

Writer Process

The code that defines the writer process is given below:

```
wait(wrt);
 WRITE INTO THE OBJECT
   signal(wrt);
```

Assignment Questions:

1. Write a C/C++ program to simulate producer-consumer problem using semaphores.
2. Write a C/C++ program to simulate reader-writer problem using semaphores.

Viva Questions:

1. What is the need for process synchronization?
2. Define a semaphore?
3. Define producer-consumer problem?

Lab 7

SIMULATE ALGORITHM FOR DEADLOCK PREVENTION

1. Deadlock Prevention

We can prevent Deadlock by eliminating any of the above four conditions.

Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

Eliminate Hold and wait

i. Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. For example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.
ii. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.

Eliminate No Preemption

Preempt resources from the process when resources required by other high priorityprocesses.

Eliminate Circular Wait

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing order of numbering.
For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 willbe granted.

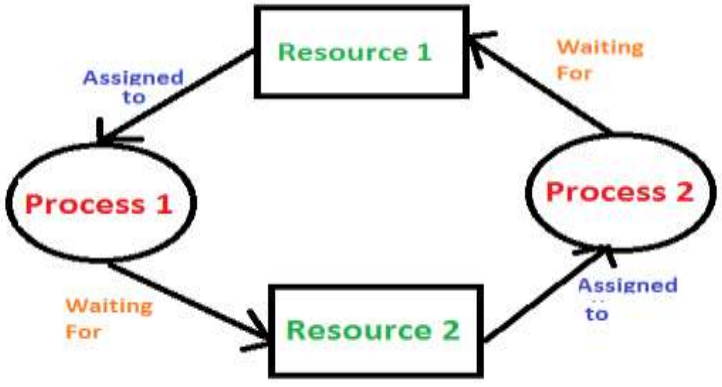
ALGORITHM:

1. Start the program
2. Attacking mutex condition: never grant exclusive access. But this may not be possible forseveral resources.
3. Attacking preemption: not something you want to do.

4. Attacking hold and wait condition: make a process hold at the most 1 resource.
5. At a time. Make all the requests at the beginning. Nothing policy. If you feel, retry.
6. Attacking circular wait: Order all the resources. Make sure that the requests are issued inthe
7. Correct order so that there are no cycles present in the resource graph. Resourcesnumbered 1 ... n.
8. Resources can be requested only in increasing
9. Order. i.e. you cannot request a resource whose no is less than any you may be holding
10. Stop

2. Deadlock Detection

i. **If resources have a single instance –**
In this case for Deadlock detection, we can run an algorithm to check for the cycle in the Resource Allocation Graph. The presence of a cycle in the graph is a sufficient condition for deadlock.



ii. In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1 → P1 → R2 → P2. So, Deadlock is Confirmed.

iii. **If there are multiple instances of resources –**
Detection of the cycle is necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

Assignment Questions:

1. Write a C/C++ program to simulate deadlock detection.
2. Write a C/C++ program to simulate deadlock prevention.

Viva Questions:

1. Define resource. Give examples.
2. What is deadlock?
3. Differentiate between deadlock avoidance and deadlock prevention?

Lab 8

1. Deadlock Avoidance

Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm

Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Inputs to Banker's Algorithm:

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition:

2. If the request made by the process is less than equal to max need to that process.
3. If the request made by the process is less than equal to the freely available resource in the system.

Example:

Total resources in system:-

A B C D

6 5 7 6

Available system resources are:-

A B C D

3 1 1 2

Processes (currently allocated resources):-

A B C D

P1 1 2 2 1

P2 1 0 3 3

P3 1 2 1 0

Processes (maximum resources):-

A B C D

P1 3 3 2 2

P2 1 2 3 4

P3 1 3 5 0

Need = maximum resources - currently allocated resources.

Processes (need resources):-

	A	B	C	D
P1	2	1	0	1
P2	0	2	0	1
P3	0	1	4	0

Note: Deadlock prevention is more strict than Deadlock Avoidance.

4. Deadlock Recovery

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is a time and space-consuming process. Real-time operating systems use Deadlock recovery.

1. **Killing the process –**
   Killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till the system recovers from deadlock. Killing all the processes one by one helps a system to break circular wait condition.

2. **Resource Pre-emption –**
   Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

3. **Ostrich Algorithm –**
   Just ignore the deadlock.

Assignment Question:

1. Write a C/C++ program to simulate Bankers algorithm for the purpose of deadlock avoidance.

Viva Questions:

1. What are the conditions to be satisfied for the deadlock to occur?

Lab 9

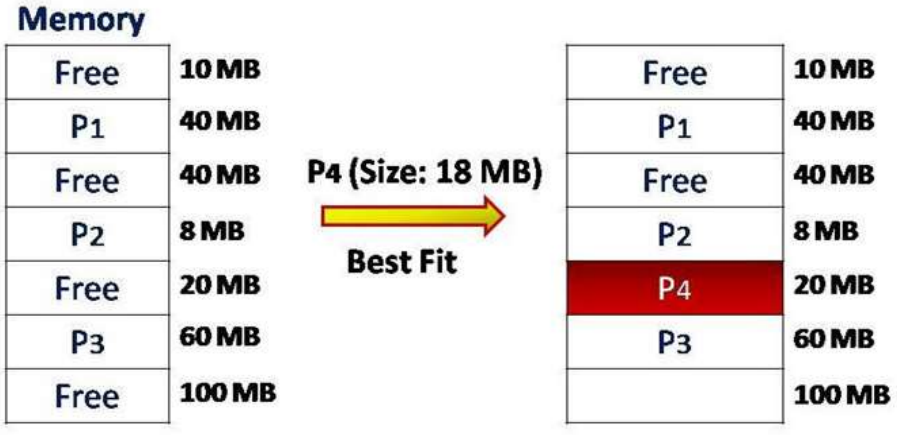
Implementation of Memory Management Algorithm: (i) Best Fit, (ii) Worst Fit and (iii) Next Fit

(i) Best Fit Algorithm

The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process.

This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate.

It then tries to find a hole which is close to actual process size needed.



Algorithm:

1. Input memory blocks and processes with sizes.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
4. If not then leave that process and keep checking the further processes.

(ii) Worst Fit Algorithm

Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory.

If a large process comes at a later stage, then memory will not have space to accommodate it.

Algorithm:

1. Input memory blocks and processes with sizes.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
4. If not then leave that process and keep checking the further processes.

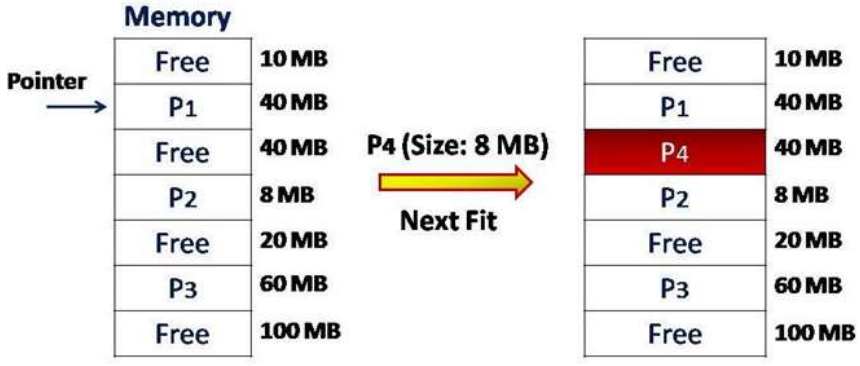
(iii) Next Fit algorithm

Next fit is a modified version of first fit. It begins as first fit to find a free partition.

When called next time it starts searching from where it left off, not from the beginning.

Algorithm:

1. Input the number of memory blocks and their sizes and initializes all the blocks as free.
2. Input the number of processes and their sizes.
3. Start by picking each process and check if it can be assigned to the current block, if yes, allocate it the required memory and check for next process but from the block where we left not from starting.
4. If the current block size is smaller then keep checking the further blocks.



Assignment Questions:

1. Write a C/C++ program to implement Memory Management concept using the following algorithms.
   a) Best fit
   b) Worst fit
   c) First fit

Viva Questions:

1. What is Memory Management?
2. Why Use Memory Management?
3. Define Best fit and its advantage?
4. What are the advantages of overlays allocation storage?

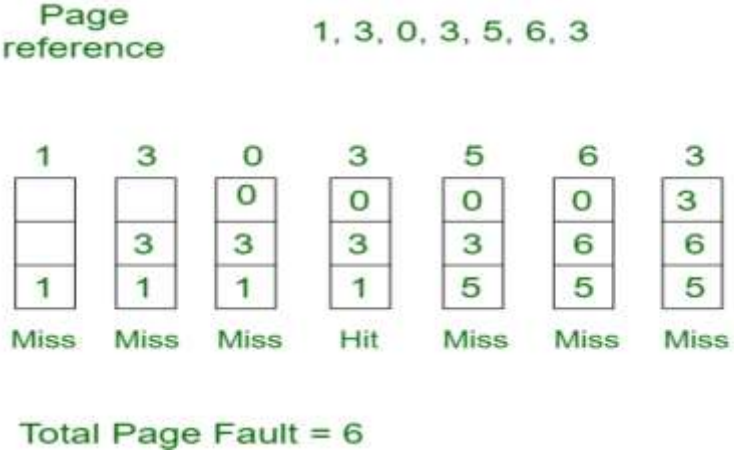
Lab 10

SIMULATE PAGE REPLACEMENT ALGORITHMS: (i) FIFO, (ii) LRU and (iii) Optimal

i. First In First Out (FIFO) –

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced, page in the front of the queue is selected for removal.

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6 and 3, with 3 page frames. Find numberof page faults.



Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3Page Faults.**
When 3 comes, it is already in memory so —> **0 Page Faults.**
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1Page Fault.**
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 PageFault.**
Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

Belady's anomaly – Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2,
1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 pagefaults.

ALGORITHM:

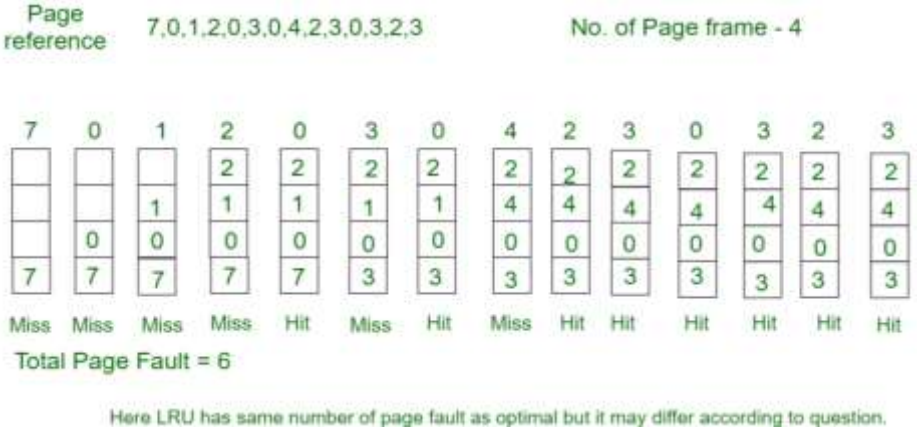
1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames in First in first out order.
7. Display the number of page faults.
8. Stop the program

ii. Least Recently Used (LRU)

AIM: To Simulate LRU page replacement algorithms

In this algorithm page will be replaced which is least recently used.

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 and 3 with 4 pageframes. Find number of page faults.



Here LRU has same number of page fault as optimal but it may differ according to question.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
When 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

ALGORITHM:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that has not been used for thelongest period of time.

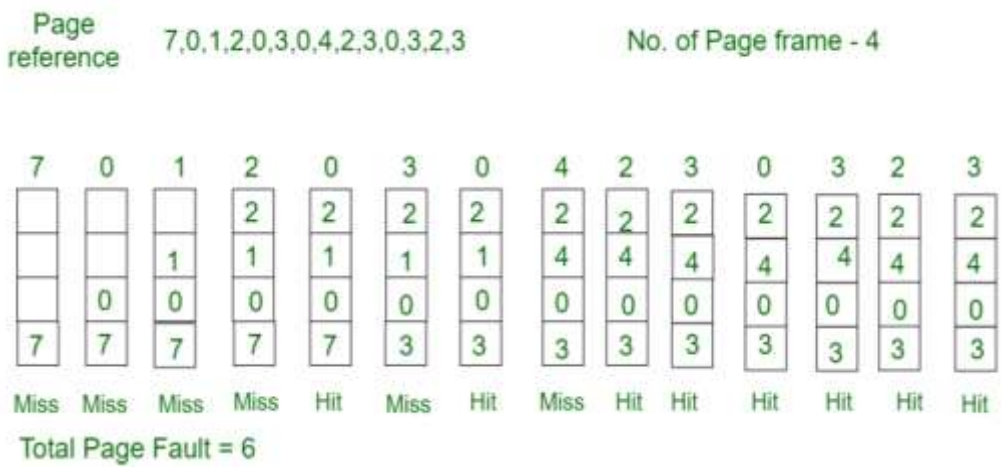
7. Display the number of page faults.
8. Stop

iii. OPTIMAL Algorithm

AIM: To create program for optimal page replacement algorithms.

In this algorithm, pages are replaced which would not be used for the longest duration oftime in the future.

Example: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, and 3 with 4 page frame.Find number of page fault.



Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
When 3 came it will take the place of 7 because it is not used for the longest duration of timein the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**
4 will takes place of 1 —> **1 Page Fault.**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.
Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmarkso that other replacement algorithms can be analyzed against it.

ALGORITHM:

1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. Stop the program

Assignment Questions:

1. Write a C/C++ program to simulate following page replacement algorithms
   a) FIFO
   b) LRU
   c) Optimal

Input Sample:

Page frames = 3

Number of pages reference string contains = 20

Page reference string 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

Viva Questions:

1. What is meant by page fault?
2. What is meant by paging?
3. What is page hit and page fault rate?
4. Which one is the best replacement algorithm?

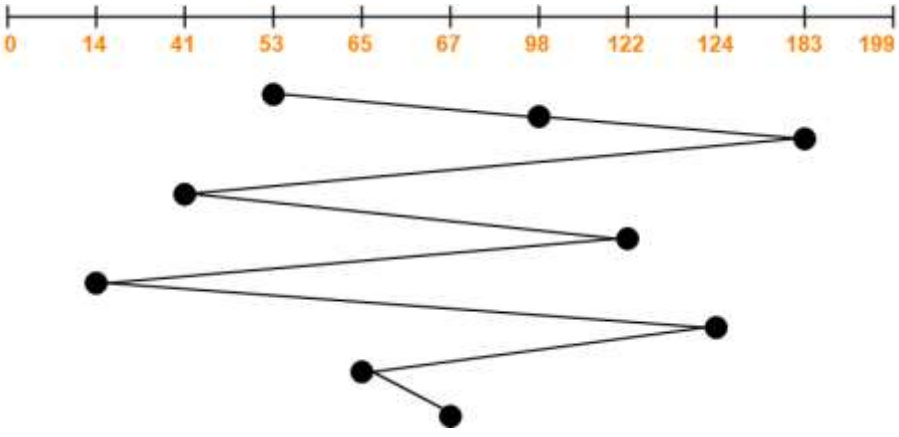
Lab 11

Implementation of Disk Scheduling Using FCFS, SCAN and C-SCAN algorithm

(i) First Come First Serve (FCFS)

This algorithm entertains requests in the order they arrive in the disk queue.

Example: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53.



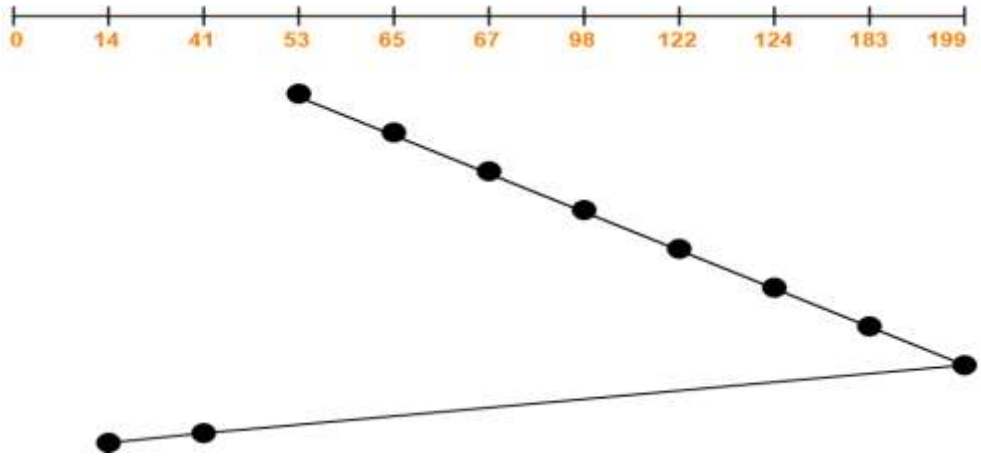
Algorithm

1. Start the program.
2. Mark the 'head' as the initial position of disk head.
3. Let request array represent an array storing indexes of track that have been requested in ascending order of their time of arrival.
4. One by one take the tracks in default order and calculate the absolute distance of the track from the head.
5. Increment the total seeks count with this distance.
6. New head position is currently serviced track position.
7. Go to step no. 3 until all track in request array have not been serviced.

(ii) SCAN Algorithm

This algorithm scans all the cylinders of the disk back and forth.
Head start from one end of the disk and move towards the other end servicing all the request in between.
After reaching the other end, head reverses its direction and move towards the starting end servicing all the requests in between.
The same process repeats.

Example: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67.The head is initially at cylinder number 53.



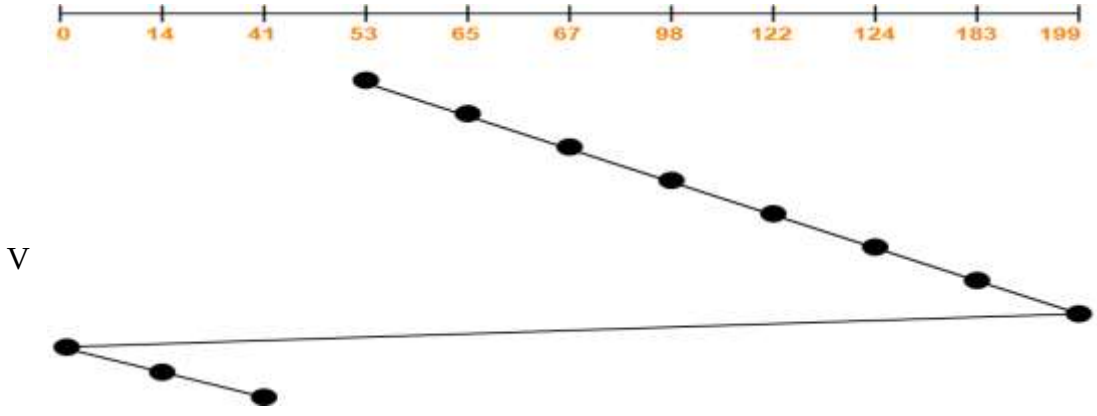
Algorithm:

1. Start the program.
2. Mark the 'head' as the initial position of disk head.
3. Let request array represent an array storing indexes of track that have been requested in ascending order of their time of arrival.
4. Let direction represents whether the head is moving towards left or right.
5. In the direction in which head is moving service all tracks one by one.
6. Calculate the absolute distance of the track from the head.
7. Increments the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 5 until we reach at one of the ends of the disk.
10. If reach at the end of the disk reverse the direction and go to step 4 until all tracks in request array have not been serviced.

(iii) Circular- SCAN (C-SCAN) Algorithm

Circular-SCAN Algorithm is an improved version of the SCAN.
Head starts from one end of the disk and move towards the other end servicing all the requests in between.
After reaching the other end, head reverses its direction.
It then returns to the starting end without servicing any request in between.
The same process repeats.

Example: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53.



Algorithm:

1. Start the program.
2. Mark the 'head' as the initial position of disk head.
3. Let request array represent an array storing indexes of track that have been requested in ascending order of their time of arrival.
4. The head services only in the right direction from 0 to the size of the disk.
5. While moving in the left directions do not service any of the tracks.
6. When we reach the beginning (left end) reverse the direction.
7. While moving in the right direction it services all tracks one by one.
8. While moving in the right directions calculate the absolute distance of the track from the head.
9. Increment the total seeks count with this distance.
10. Currently serviced track position now becomes the new head position.
11. Go to step 8 until we reach the right end of the disk.
12. If we reach the right end of the disk reverse the direction and go to step 5 until all tracks in the request array have not been serviced.

Assignment Questions:

1. Write a C/C++ program to simulate following disk scheduling algorithms
   a) FCFS
   b) SCAN
   c) C-SCAN

Input Sample

Enter Number of Tracks: 10

Enter Track Position: 50 55 18 40 60 120 67 80 91 22

Viva Questions:

1. What are the terms – disk seek time, disk access time and rotational latency?
2. What is the advantage of C-SCAN algorithm over SCAN algorithm?

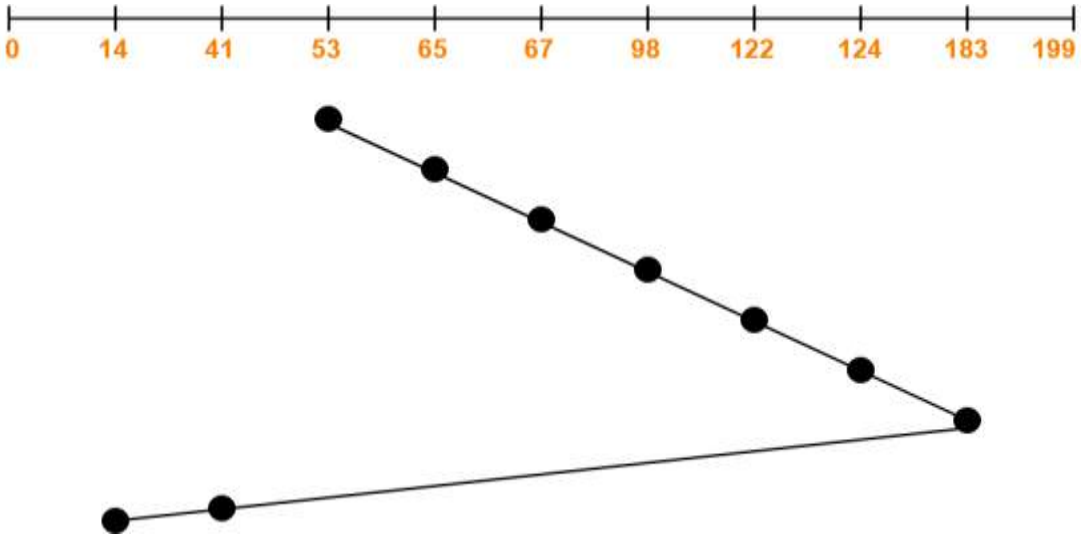
Lab 12

Implementation of Disk Scheduling Using LOOK, C-LOOK and SSTF algorithm

(i) LOOK Algorithm

LOOK Algorithm is an improved version of the SCAN Algorithm.
Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
After reaching the last request at the other end, head reverses its direction.
It then returns to the first request at the starting end servicing all the requests in between.
The same process repeats.

Example: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67.The head is initially at cylinder number 53.



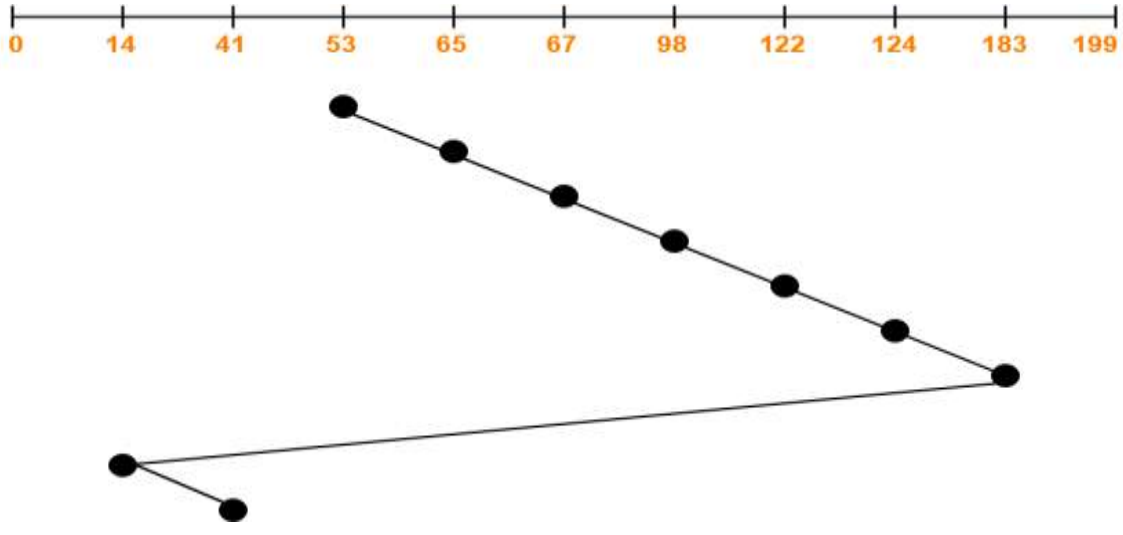
Algorithm

1. Start the program.
2. Mark the ‘head’ as the initial position of disk head.
3. Let request array represent an array storing indexes of track that have been requested in ascending order of their time of arrival.
4. The initial direction in which head is moving is given and it services in the same direction.
5. The head services all the requests one by one in the direction head is moving.
6. The head continues to move in the same direction until the entire request in this direction are finished.
7. While moving in this direction calculates the absolute distance of the track from the head.
8. Increment the total seeks count with this distance.
9. Currently serviced track position now becomes the new head position.
10. Go to step 7 until we reach at last request in this direction.
11. If we reach where no requests are needed to be serviced in this direction reverse the direction and go to step 5 until all tracks in request array have not been serviced.

(ii) C-LOOK Algorithm

Circular-LOOK Algorithm is an improved version of the LOOK Algorithm.
Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
After reaching the last request at the other end, head reverses its direction.
It then returns to the first request at the starting end without servicing any request in between.
The same process repeats.

Example: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The head is initially at cylinder number 53.



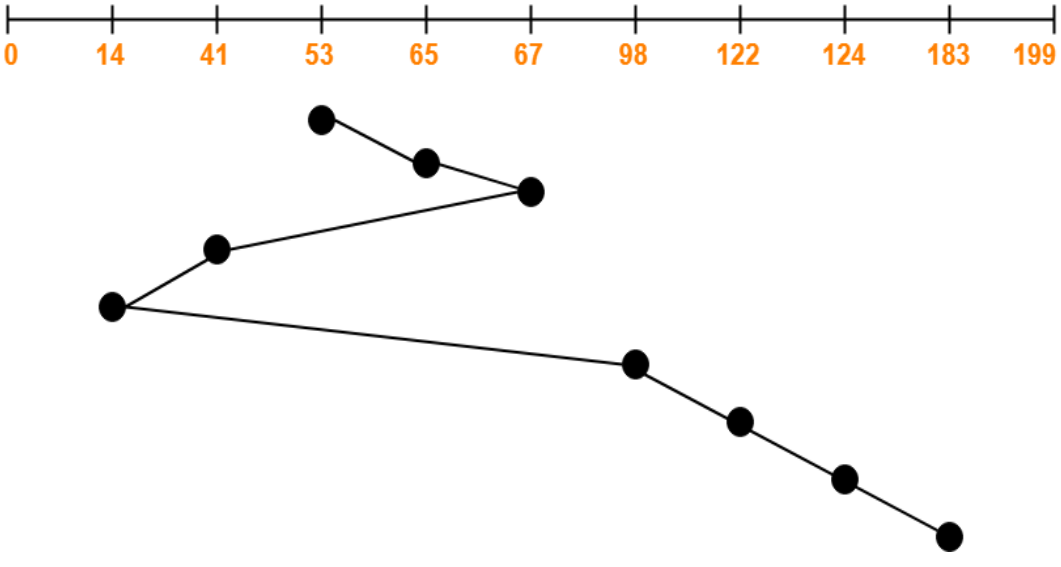
Algorithm:

1. Start the program.
2. Mark the 'head' as the initial position of disk head.
3. Let request array represent an array storing indexes of track that have been requested in ascending order of their time of arrival.
4. The initial direction in which the head is moving is given and it services in the same direction.
5. The head services all the requests one by one in the direction it is moving.
6. The head continues to move in the same direction until all the requests in this direction have been serviced.
7. While moving in this direction, calculate the absolute distance of the tracks from the head.
8. Increment the total seeks count with this distance.
9. Currently serviced track position now becomes the new head position.
10. Go to step 5 until we reach the last request in this direction.
11. If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
12. Reverse the direction and go to step 3 until all the requests have not been serviced.

(iii) Shortest Seek Time First (SSTF)

SSTF stands for Shortest Seek Time First.
This algorithm services that request next which requires least number of head movements from its current position regardless of the direction.
It breaks the tie in the direction of head movement.

Exapmle: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67.The head is initially at cylinder number 53.



Algorithm:

1. Start the program.
2. Mark the 'head' as the initial position of disk head.
3. Let request array represent an array storing indexes of track that have been requested in ascending order of their time of arrival.
4. Find the positive distance of all tracks in the request array from head.
5. Find a track from requested array which has not been accessed/serviced yet and has minimum distance from head.
6. Increments the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 4 until all tracks in request array have not been serviced.

Assignment 12:

1. Write a C/C++ program to simulate following disk scheduling algorithms
   a) LOOK
   b) C-LOOK
   c) SSTF

Input Sample:

Enter Number of Tracks: 10

Enter Track Position: 50 55 18 40 60 120 67 80 91 22

Viva Questions:

1. Which disk scheduling algorithm has highest rotational latency? Why?

Project