its own vector clock and the value in the vector in the received message (for every element).

Implementation Rules [IR]:-

[IR1]: Vector clock vci is incremented between any two successive events in process Pi

$$VCi[i] = VCi[i] + d,\ d>0.$$

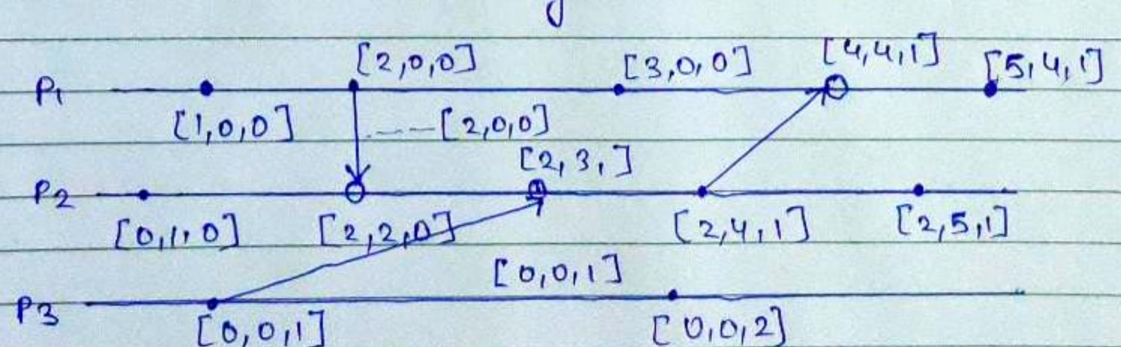
[IR2]: If event a is the sending event of message ~~o~~ m by process Pi then m is assigned a vector timestamp $tm = VCi(a)$ On receiving the same msg m by process Pj.

VCj is updated as,

$\forall K, VCj[K] = max\ [VCj[K], tm[K])$
K is no. of process.

Example:-

Consider a process (P) with a vector size N for each process: the above set of rules mentioned are to be executed by the vector clock:



● = EVENT EXECUTING RULE1

o = EVENT EXECUTING RULE 2

⟶
TIME

PRACTICAL NO. 02

AIM:- To implement vector clock in Distributed System.

THEORY:- Vector clock is an algorithm that generates partial ordering of events and detects causality violations in a distributed system.

- These clocks expand on Scalar Time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system.

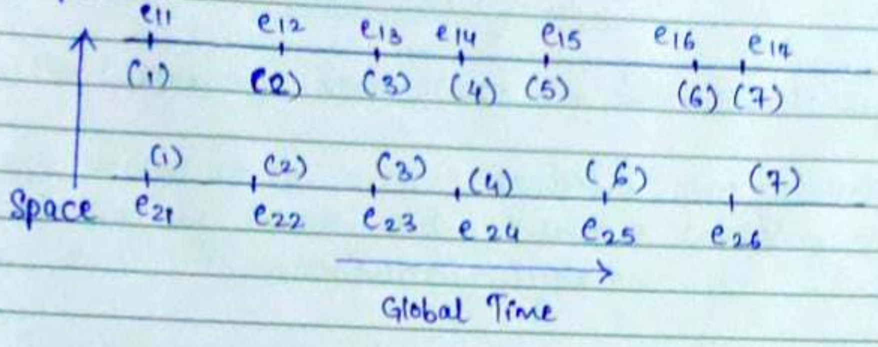
- It essentially captures all the causal relationships. This algorithm helps us label every process with a vector (a list of integers) with an integer for each local clock of every process within the system. So for N given processes, there will be vector/array of size N.

ALGORITHM:-

- Initially, all the clocks are set to zero.
- Every time, an internal event occurs in a process, the value of the processes's logical clock in the vector is incremented by 1.
- Also, every time a process sends a message, the value of the processes's logical clock in the vector is incremented by 1.

- Every time, a process receives a message, the value of the processes's logical clock and in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in

Example:-



- Take the string value as 1, since it is the 1st event and there is no incoming value at the starting point:
  - $e_{11} = 1$
  - $e_{21} = 1$
- The value of the next point will go on increasing by d (d=1), if there is no incoming value i.e., to follow [IR1].
  - $e_{12} = e_{11} + d = 1+1 = 2$
  - $e_{13} = e_{12} + d = 2+1 = 3$
  - $e_{14} = e_{13} + d = 3+1 = 4$
  - $e_{15} = e_{14} + d = 4+1 = 5$
  - $e_{16} = e_{15} + d = 5+1 = 6$
  - $e_{22} = e_{21} + d = 1+1 = 2$
  - $e_{24} = e_{23} + d = 3+1 = 4$
  - $e_{26} = e_{25} + d = 6+1 = 7$
- when there will be incoming value, then follow [IR2] i.e., take the maximum value between C_j and $T_m + d$.
  - $e_{17} = max(7,5) = 7$, [$e_{16} + d = 6+1 = 7$, $e_{24} + d = 4+1 = 5$, maximum among 7 and 5 is 7]
  - $e_{23} = max(3,3) = 3$, [$e_{22} + d = 2+1 = 3$, $e_{12} + d = 2+1 = 3$, maximum among 3and 3 is 3]
  - $e_{25} = max(5,6) = 6$, [$e_{24} + 1 = 4+1 = 5$, $e_{15} + d = 5+1 = 6$, maximum among 5 & 6 is 6]

Program and output :- (Attached)

Result :- The lamport's logical clock is implemented succesfully.

j^{th} element is Ci[j] and contains Pi's latest value for the current time in process Pj.

- d: drift time, generally d is 1.

Implementation Rules [IR]:-

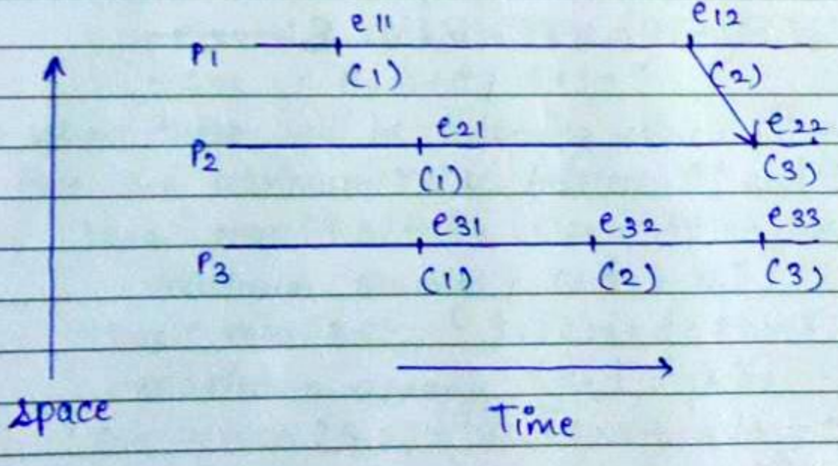
- [IR1]: If a → b ['a' happened before 'b' within the same process then, Ci(b) = Ci(a) + d

- [IR2]: Cj = max (Cj, tm+d) [If there's more number of processes, then tm = value of Ci(a), Cj = max value between Cj and tm+d]

Limitation:-

- In case of [IR1], if a → b, then C(a) < C(b) → true.

- In case of [IR2], if a → b, then C(a) < C(b) → May be true or may not be true.



PRACTICAL NO. 01

AIM: To implement Lamport's Logical clock.

THEORY: Lamport's logical clock was created by Leslie Lamport. It is a procedure to determine the order of events occuring. It provides a basis for the more advanced vector clock algorithm.

- Due to the absence of a Global clock in a Distributed operating system, Lamport logical clock is needed.

ALGORITHM:

- Happened before relation ($\rightarrow$): $a \rightarrow b$, means 'a' happened before 'b'.
- Logical clock : The criteria for the logical clocks are :-
- [C1] : $C_i(a) < C_i(b)$, [$C_i \rightarrow$ Logical clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process.]
- [C2] : $C_i(a) < C_j(b)$, [Clock value of $C_i(a)$ is less than $C_j(b)$].

Reference :-

- Process : P_i
- Event : E_{ij}, where i is the process in number and j : jth event in the ith process.
- t_m : vector time span for message m.

- This example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

- Vector clock algorithms are used in distributed systems to provide a causally consistent ordering of events but the entire vector is sent to each process for every message sent, in order to keep the vector clock in sync.

PROGRAM & OUTPUT :- (Attached)

RESULT :- The vector clock is implemented successfully.

Practical No: 1

Aim → To demonstrate classical cipher. Encrypt the given Text message with the help of following algorithm.

1) Replace each alphabet with its equivalant 7-bit ASCII Code

2) Swap first four bites with last four bits for each alphabets.

3) Write a hexadecimal equivalent for every four bit.

Theory → Cryptography is a method of Protecting information and communication through the use of codes. So that only those for whom the information is intended can read & process it.

Cryptography refers to secure information & Communication techniques derived from mathamatical concepts and a set of rule-based Calcutions called algorithms to transform messages in ways that they are hard to decipher.

Plain Text → Information that can be directly read by humans or a machine.

Ciper Text → The encrypted Text.

Encryption → The process of converting the plain Text to Cipher Text.

Decryption → The process of reversing back ciphertext to Plaintext.

```
    e.append(s[i][4:])
   print(e)
 def binToHexa(n):
          num = int(n,2)
   hex_num = hex(num)
       return(hex_num)
 print("Hexadecimal Equivalent: ")
    for i in range(len(e)):
         print(binToHexa(e[i]), end=" ")
```

Reasult ->

The program to demonstrate classical cipher executed succesfully.

a) Replacing each alphabet with its equvalent 7 bit ASCII Code

b) swapping first four bits with last four bits for each alphabets.

c) Obtaining a hexadecimal equivalent for every four bits is implemented succesfully.

```
Program —> num=input("Enter the string")
ascii_values=[ord(character) for character in num]
print("Equivalant ascii value of string:",acii_values)
 str1 = []; f=[]; e=[]; sw=[]; s=[]
   def decimalToBinary(n):
       return bin(n).replace("b","")
 For i in num:
        Num:ord(i)
        ch= decimal to Binary (Num)
         str1.append(ch)
 print("Before swapping:",str1)
 for j in range(len(str1)): #-> for reversing each bin of char
  for i in str1[j]
           e.insert(0,i)
         f.append(e)
           e=[]
        # print("rev:", f)
           e=[]
for i in range(len(f)); # swaping first 4 bits with last 4 bits of each character
      for j in f[i]:
            if len(e)<4;
               e.insert(0,j)
         else:
            e.insert(4,j)
         sw.append(e)
              e=[]
    print("After swapping: ", sw)
       for i in range(len(sw)):
             s.append("".join(sw[i]))
      # print(s)
          e=[]
```

Practical No -> 4.

Aim -> Write a program to implement Ricart's Agarwala Algorithm.

Theory -> Richart - Agarwala Algorithm is an algorithm for Mutual exclusion in a distibuted system Proposed by ' Gleen Ricart and Ashok Agarwala". This algorithm is an extension and optimization of Lamports Distibuted Mutual Excelusion Algorithm. like Lamports algorithm, it also follow permision based approch to ensure Mutual Exelusion.

Algorithms ->

① To enter the Critical section (CS) :- • When a site Si wants to enter the critical section, it send a timestamped REQUEST section/message to all other sites.

• When a site Sj recives a REQUEST message to site Si if and only if -

① site Sj is neither requesting nor Currently executing the critical section

② In case site Sj is requesting, the timestamp of site Si's request is smaller than its own request.

Otherwise the request is deferred by site Sj

② To Execute the Critical section ->

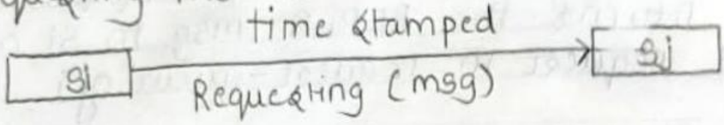
Site Si enters the critical section if it has recived the REPLY message from all other sites.

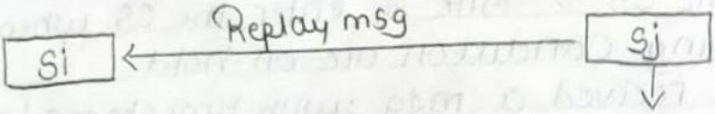
Practical NO -> 4

Aim -> Write a program to implement Ricart Agarwala mutu exclusion algorithm.

Ricart - Agarwala Mutual Exclusion Algorithms

Requesting the CS ->

①


②


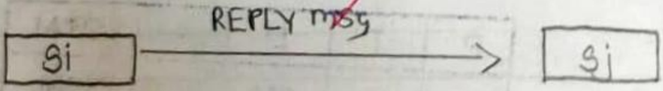
① If Sj neither requesting

② Nor Executing

③ If requesting but long timestamp -> defined.

2) Executing the CS ->
Si enter the CS after recived REPLY msg from all sites.

3) Releasing the CS ->



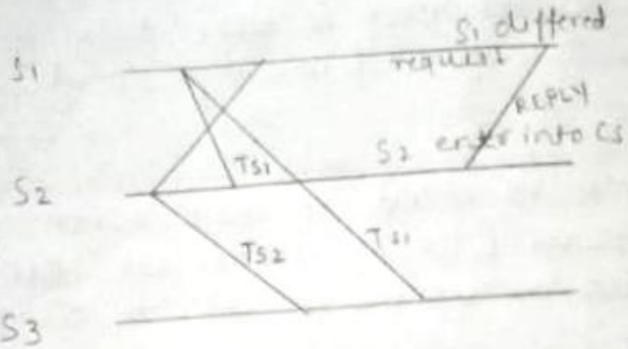
{ deferred msg (Request)

Practical No: 04

Aim:- Write a program to implement Ricard's Agrawala mutual Exclusion algonthm

RAM:- Optimization of the lamport algonthm

Ts: Timestamp



condition

① Executing the CS
Si enter the CS after received REPLY msg from all sites

③ Releasing the CS P
$S_i \xrightarrow[msg]{REPLY} S_j$

server by the client's operating system

- The message is passed to the server stub by the server operating system.
- The parameter are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

Result: Thus a program to implement client server based program using the RPC is executed successfully

Practical No :- 05 30/3/22

Aim :- Write a client server program based on RPC

Theory :- RPC :- Remote Procedure call is a software communication protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network detail's

Server client model :- The client server model is distributed application structure that partitions task or workload between the providers of a resource or service called server's and a service requesters called clients

A client has a request message that the RPC translates and sends to the server this request may be a procedure or a function call to a remote server when the server receives the request, it sends the required response back to the client The client is blocked while the server is processing the call and only resumed execution after the server is finished

The sequence of events in a ~~resum~~ remote procedure call are given below

* The client stub is called by the client
* The client stub makes a system call to the message to the server and puts the parameters in the message
* The message is sent from the client to the

the sites in the request set (R) and places the request in the request - queue [(tsi, i)] is the timestamp of the request

b) When a site S_j receives the request (tsi, i) msg from site S_i, It return the REPLY msg to S_i and places S_i's request in request - queue of j

Executing the critical section: Site S_i enters the cs when the two following cond^n are on hold

[L1]: S_i has recieved a msg with timestamp lauger than (ts, i] from all the sites
[L2]: S_i is request is at to of the queue

Releasing critical section: Site S_i, upon exitiong the C's removes request from the top of request - queue and send the timestamp release msg to all sites in request When sites S_j recieves a release msg from site S_i, it remores S_j's request from it's queue.

Result: The program to implement lamport's mutual exclusion algorithm is studied and executed successfully

Practical No : 04 25/3/22

Aim : Write a program to implement Ricart's Agrawala mutual exclusion algorithm

Theory : Ricart's Agrawala mutual exclusion algorithm is the optimization of lamport's algorithm that dispreses with RELEASE message by clearly merging them with REPLY message in this algorithm for all $1 <= i <= N$: $R_i = \{S_1, S_2, \ldots S_N\}$

Implementation : Requesting the critical section

1] When a site S_i wants to enter the CS, it sends a timestamped REQUEST message to all the in its request set

2] When a site S_j receives the REQUEST msg from site S_i, it send a REPLY message to site S_i if the site S_j is neither requesting nor executing the CS or if the site S_j is requesting and S_j's own request's timestamp. The request is deferred otherwise Executing the critical section

3] Site S_i enters the CS after it has received REPLY message from all the sites in its request set Releasing the critical section

4] When site S_j, exits the CS, it sends REPLY message to all the deferred requests A sit's REPLY message are blocked only by sites that are requesting the CS with higher priority There when a site sends out REPLY message to all the deferred requests The site sends out REPLY message to all the deffered request The site with the next highest priority request recieves the last needed REPKY message and enters the CS. The execution of CS request

Practical No :- 03 25/3/22

Aim : Write a program to implement lamport's mutual exclusion algorithm.

Theory :- A mutual exclusion is a program object that present simultaneously access the phased resources. Their concept is used in concurrent programming with a critical section a piece of code in which process accessed.

Requirments of mutual exclusion algorithm

1] freedom from deadlock
2] freedom from starvation
3] fairness
4] fault tolerant

classification of mutual exclusion algorithm

1] Token based algorithm
2] Non - Token based algorithm

critical section - When more than one process access a same code segment that segment is known as critical section

Basic idea of non - token based algorithm

The sites have there states

1] Requesting the critical section
2] Executing the critical section
3] Releasing the critical section

lamport's mutual Exclusion:

Requesting critical section

a] When site Si wants to enter the CS it sends in this algorithm is always in the order of their timestamps.

Result: Thus a program to implement Ricart's Agrawala mutual exclusion algorithm is executed successfully.

Practical No. 7

Aim :- WAP to implement Chandy's Misra's Haas deadlock detection algorithm.

Theory: Chandy's - Misra - Haas's distributed deadlock detection algo is an edge chasing algo to detect deadlock in distributed system.

In edg chasing algo, a special msg called probe is used in deadlock detection. A probe is a triplet (i,j,k) which denotes that process P_i has initiated the deadlock detection & the msg is being sent by the home site of process P_j to the home site of process P_k.

Algorithm :

Process of sending probe :

1. If process P_i is locally dependent on itself then declare a deadlock.
2. Else for all P_j and P_k check following condition:
   - a) Process P_i is locally dependent on process P_j
   - b) Process P_j is waiting on process P_k.
   - c) Process P_j and process P_k are on different sites.

   If all of the above conditions are true, send probe(i,j,k) to the home site of process P_k.

On the receipt of prob (i,j,k) at home site of process P_k:

1. Process P_k checks the following conditions:
   - a) Process P_k is blocked
   - b) $\text{dependent}_k[i]$ is false
   - c) Process P_k has not replied to all requests of process P_j.

   If all of the above conditions are found to be true then :-

1. Set dependentk [i] to true.
2. Now, If k == i then, declare the Pi is deadlock.
3. Else for all Pm & Pn check following conditions:
- a) Process Pk is locally dependent on process Pm and
- b) Process Pm is waiting upon process Pn and
- c) Process Pm and process Pn are on different sites.

4. Send probe (i, m, n) to the home site of process Pn if above conditions satisfy.

Thus, the probe msg travels along the edges of transaction wait for (TWF) graph and when the probe msg return to its initiating process then it is said that deadlock has been detected.

Performance :

- Algorithm require at most exchange of $m(n-1)/2$ msg to detect deadlock. Here, m is no. of process and n is the number of sites.
- The delay in detecting the deadlock is $O(n)$.

Advantages :

- There is no need for special data structure & probe msg, which is very small & involves only 3 integers & a two demensional boolean array dependent is used in the deadlock detection process.
- At each site, only a little computation is required & overhead is also low.

Disadvanteges :

The main disadvantage of a distributed detection algo is that all sites may not aware of the processes involved in the deadlock this makes resolution difficult. Also proof of correction of the algo is difficult.

Result : Hence, we have successfully implemented Chandy - Misra's - Haas's deadlock detection algo.

Practical No. 8

Aim: WAP to implement sliding window protocal.

Theory :

Window Sliding Technique :-

WST is a computational technique which aims to reduce the use of nested loop and replace it with a single loop, thereby reducing the time complexity.

What is sliding Window?

Consider a long chain connected together. Suppose you want to apply oil in the complete chain with your hands, without pouring the oil from above.

One way to do so it to is to :

- pick some oil,
- apply onto a section of chain,
- then again pick some oil.
- then apply it to the next section where oil is not applied yet.
- and so on till the complete chain is oiled.

The second way is known as the Sliding window technique and the portion which is slided from one end to end, is known as sliding window.

Prerequisite to use sliding window technique.

The use of sliding window technique can be done in a very specific scenario, where the size of window for computation is fixed throughout the complete nested loop. Only then the time complexity can be reduced.

Applying Sliding window technique :

1. We compute the sum of first k elements out of n terms using a linear loop & store the sum in variable window-sum.

2. Then we will graze linearly over the array till it reaches the end & simultaneously keep track of maximum sum.

3. To get the current sum of block of k elements just subtract the first element from the previous block & add the last element of the current block.

Result: Hence, we have successfully implemented sliding window technique.

Advantages of pbft :

- **Energy efficiency** :- pBFT can achieve distributed consensus without carrying out complex mathematical computation. Zilliqa employs pBFT in combination with PoW-like complex computation round for every 100th block.

- **Transaction finality** : the transaction do not requir multiple confirmations after they have been finalized and agreed upon.

- **Low reward variance** : Every node in the netwo takes part in responding to the request by the client & hence every node can be incentivized leading to low variance in rewarding the nodes that help in decision making.

Result : Hence, we have successfully learned about solution of Byzantine agreement problem.

Practical No. 9

Aim: WAP for solution of Byzantine agreement problem

Theory :

Byzantine Agreement problem :

~~In~~ In the Byzantine Agreement problem, a single value, which is to be agreed on, is initialized by an arbitrary processor and all non-faulty processoes have to agree on that value.

Solution of Byzantine agreement problem :

In general, a sol^n to an agreement problem must pass three tests : termination, agreement, and validity. As application to the Byzantine General's problem, these three tests are :

1. A solution has to guarantee that all correct processes eventually reach a decision regarding the value of the order they have been given.
2. All correct processes have to decide on the same value of the order they have been given.
3. If the source process is a correct process, all processes have to decide on the value that was original given by the source process.

Type of Byzantine failures :

There are two categories of failure that can considered. One is fail-stop & other is arbitrary-node failure. Some of the arbitrary node failures are given below :

- Failure to return a result.
- Respond with an incorrect result.
- Respond with a deliberately misleading result.
- Respond with a different result to different parts of the system.

Practical No. 10

Aim:- To study real time operating system.

Theory:

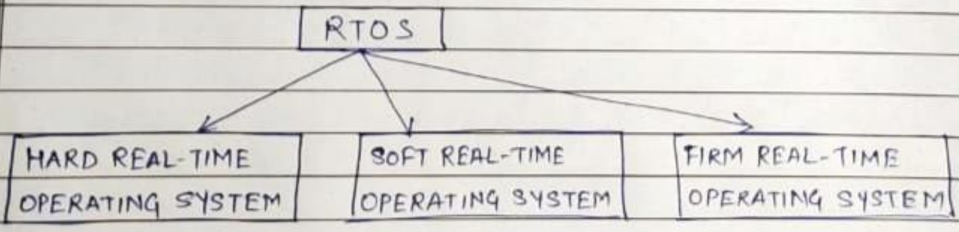
Real time operating System (RTOS)

Real time operating system(RTOS) are used in environment where a large no. of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines. Such applicat^n are industrial control, telephone switching equipment, flight control, and real-time simulations.

When an RTOS, the processing time is measured in tenths of second. This system is time-bond bound & has a fixed deadline. The processing in this type of system must occur within the specified constraints. Otherwise, this will lead to system failure.

Example of the real-time operating System: Airline traffic control system, Command control system, Airline reservation system, Heart Peacemaker, Network Multimedia systems, Robot etc.

The real-time operating system can be of 3 types -


RTOS
HARD REAL-TIME OPERATING SYSTEM
SOFT REAL-TIME OPERATING SYSTEM
FIRM REAL-TIME OPERATING SYSTEM

1. HARD Real-Time operating system :

These operating system guarantee that critical task be completed within a range of time.

For ex, a robot is hired to weld a car body. If the robot welds too early or too late, the car cannot be sold, so it is a hard real-time system that requires complete care welding by robot hardly on the time.

2. Soft real time operating system :

This operating system provides some relaxation in the time limit. For ex, - Multimedia systems, digital audio system etc. Explicit, programmer defined & controlled processes are encountered in real time system. A separate process is changed with handling a single external event. This process is activated upon occurrence of the related event signalled by an interrupt.

Multitasking operation is accomplished by scheduling processes for execution independently of each other. Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services. The processor is allocated to the highest priority processes. This type of schedule called priority-based preemptive scheduling is used by real-time system.

3. Firm Real-time Operating System :

RTOS of this type have to follow deadline as well. In spite of its small impact, missing a deadline can have unintended consequences, including a reduction in the quality of the product.

Ex. Multimedia applications.

- Advantages :

The advantages of real-time operating systems are as follows :-

1. Maximum Consumption -

Maximum utilization of devices and systems. Thus more output from all the resources.

2. Task Shifting -

Time assigned for shifting tasks in these system is very less. For ex, in order older systems, it takes about 10 microsecond. Shifting one task to another & in the latest systems, it takes 3 microseconds.

3. Focus On Application —

Focus on running applications and less importance to applications that are in the queue.

4. Real-Time Operating System In Embedded system—

Since the size of program is small, RTOS can also be embedded systems like in transport and others.

5. Error Free -

These type of systems are error - free.

6. Memory allocation -

Memory allocation is best managed in these types of systems.

Disadvantages :

The disadvantages of real-time operating systems are as follows -

1. Limited Tasks -

Very few tasks run simultaneously, and their concentration is very less on few applications to avoid errors.

2. Use Heavy System Resource -

Sometime the system resources are not so good and they are expensive as well.

3. Complex Algorithms -

The algo. are very complex & difficult for the designer to write on.

4. Device Driver And Interrupt signals -

It needs specific device drivers and interrupts signals to respond earliest to interrupts.

5. Thread Priority -

It is not good to set thread priority as these systems are very less prone to switching tasks

6. Minimum Switching - RTOS performs minimal task switching.

Result: Hence, we have successfully learned about real-time operating system.

C=2