Submitted By:

Ankit B.K. (THA076BCT006)

Ayush Batala (THA076BCT011)

Mishan Thapa Kshetri (THA076BCT019)

Nishant Uprety (THA076BCT023)

Solution of 2076 Chaitra

Q1. Define Distributed System (DS). Explain the requirements to successfully implement DS to suit to modern computing.

Ans: A distributed system is the system which consist of collection of autonomous computers, connected through a network and distributed operating system software which enables computers to co-ordinate their activities and to share the resource of the system so that users perceive the system as single, integrated computing facility. In other word, A distributed system is a collection of independent computers at networked locations such that they communicate and interact only through message passing that is viewed as a single system by its users.

Requirements for successfully implementing a Distributed System in the context of modern computing include:

a. Scalability: The system should be able to handle an increasing number of nodes, users, or tasks without a significant decrease in performance. Horizontal scalability, where new nodes can be added easily, is particularly important.

b. Reliability and Fault Tolerance: Distributed systems should be designed to continue functioning even in the presence of hardware or software failures. This involves redundancy, replication, and error recovery mechanisms.
c. Availability: The system should be available and responsive to users' requests even when some components are unavailable due to failures or maintenance.
d. Consistency and Data Integrity: Ensuring that data remains consistent and accurate across all nodes in the system is crucial. This involves maintaining data integrity during concurrent access and updates.
e. Concurrency and Synchronization: Distributed systems often involve multiple processes or threads accessing shared resources concurrently. Implementing mechanisms for proper synchronization and managing concurrent access is essential.
f. Load Balancing: Distributing the workload evenly across nodes helps prevent resource bottlenecks and ensures optimal utilization of resources.
g. Security: Distributed systems should implement strong security measures to protect data and prevent unauthorized access. This includes authentication, encryption, and access control mechanisms.
h. Transparency: Users and applications interacting with the distributed system should perceive it as a single, coherent entity rather than a collection of individual components.
i. Interoperability: Modern distributed systems often involve components built on different platforms, programming languages, or technologies. Ensuring these components can work seamlessly together is important.
j. Heterogeneity: Distributed systems can consist of diverse hardware and software components. The system should be designed to accommodate this heterogeneity and facilitate interoperability.
k. Resource Sharing: Efficiently sharing computational resources, such as CPU, memory, and storage, among different nodes is a key requirement.
l. Global State Management: In cases where the system spans across geographical regions, maintaining a consistent global state and handling potential latency challenges become crucial.

Q2. Discuss the functionalities provided by RMI Software. How is event and notification system implemented in distributed object-based communication.

Ans: RMI is the means by which objects in different processes can communicate with one another. It allows object in one process to invoke or call the methods of an object in another process. RMI uses stub and skeleton object for communication with the remote object. Functionalities provided by RMI Software are:

a. Remote Object Invocation: RMI enables a client to invoke methods on a remote object as if it were a local object. The client-side stub handles the communication details, such as marshaling parameters and sending requests over the network, while the server-side skeleton handles receiving the request, invoking the method on the actual object, and returning the result.
b. Security: RMI provides security mechanisms to control access to remote objects and prevent unauthorized access. This includes authentication and authorization features.
c. Registry Service: RMI uses a registry service to bind remote objects to names so that clients can look up and locate these objects. The registry acts as a central repository for object references.
d. Asynchronous Communication: While RMI primarily supports synchronous remote method invocation, developers can implement asynchronous communication patterns by using techniques such as callbacks.

An event and notification system is a crucial for allowing components in a distributed system to communicate and react to changes or events. It can be implemented in the following ways:

a. Event Producer Registration: Components that generate events (event producers) register themselves with a central event management system. In the context of RMI, these event producers could be remote objects that provide event-related methods.
b. Event Consumer Subscription: Components interested in receiving notifications about specific events (event consumers) subscribe to the event management system. Again, in an RMI context, these consumers could be remote objects that provide callback methods.

c. Event Notification: When an event occurs, the event producer invokes a method on the event management system. The event management system maintains a list of subscribed consumers for each event type.
d. Callback Invocation: The event management system looks up the subscribed consumers and invokes callback methods on these remote objects using RMI. These callback methods notify the consumers about the event.
e. Data Passing: If the event includes data, it can be passed as parameters to the callback methods. RMI's serialization mechanisms handle the data transfer seamlessly.
f. Error Handling and Retry: Robust event systems should handle network failures, retries, and potential errors in delivering notifications to remote consumers.
g. Unsubscription: Consumers can unsubscribe from events when they are no longer interested, and event producers can deregister from the event system if they are no longer active.
h. Security: Ensure proper security measures are in place to prevent unauthorized access to events and to protect the integrity of the event system.

Q.3 Explain the distributed file system? Explain the principle operation of any one modern distributed file system?

Ans: A distributed file system (DFS) is a file system that spans across multiple file servers or multiple locations, that are situated in different physical places. In DFS, files are accessible just as if they were stored locally, from any device and from anywhere on the network.

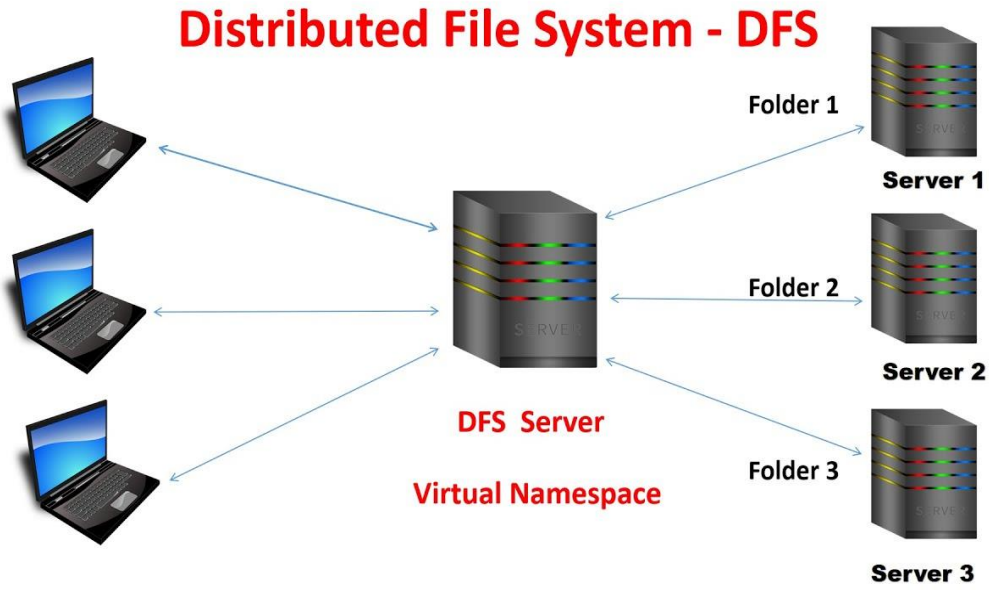


Fig: Distributed File System (DFS)

The Sun Network File System (NFS) is one of the earliest and most successful network file systems. It was developed by Sun Microsystems and designed to provide transparent access to remote files and directories across a network, particularly for diskless workstations. NFS allows users and applications to access and manipulate files on remote servers as if they were local files. The focus of NFS is on transparency, robustness, and performance.

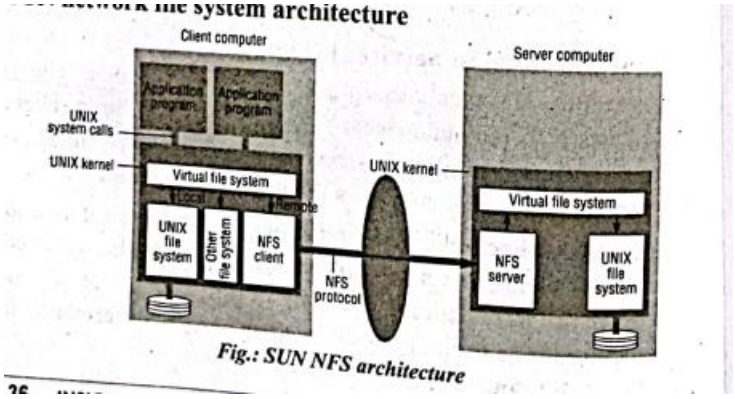


Fig: Sun NFS Architecture

The principle operations of Sun NFS are described below:

a. Search for File within Directory:
    - Clients initiate a request to search for a specific file within a directory on the NFS server.
    - The server processes the request and searches the specified directory for the requested file.
    - The result is sent back to the client, indicating whether the file was found and its location if found.
b. Read a Set of Directory Entries:
    - Clients request a list of directory entries (filenames and metadata) from a specific directory on the NFS server.
    - The server retrieves the requested directory entries and sends the list back to the client.
    - This operation allows clients to browse the contents of a remote directory.
c. Manipulate Links and Directories:
    - Clients can create, delete, or manipulate symbolic links, hard links, and directories on the NFS server.
    - These operations enable clients to manage the organization of files and directories on the remote server.
d. Read File Attribute:
    - Clients request the attributes (metadata) of a specific file on the NFS server, such as permissions, owner, size, and timestamps.
    - The server retrieves the requested attributes and sends them back to the client.
e. Write File Attribute:
    - Clients can modify the attributes of a specific file on the NFS server, such as changing permissions or timestamps.
    - The server updates the attributes based on the client's request.
f. Read File Data:
    - Clients initiate a request to read the contents of a specific file on the NFS server.
    - The server retrieves the requested data and sends it back to the client.
    - This operation allows clients to access the content of remote files.

g. Write File Data:
    - Clients can write data to a specific file on the NFS server.
    - The server stores the incoming data in the specified file.
    - This operation allows clients to modify the content of remote files.

Q.4 What is the issue in Lamport's timestamp? How do you avoid the issue? Explain with your alternate algorithm.

Ans: Lamport's timestamp is the procedure to determine the order of event occurring. The issues in Lamport's timestamp algorithm are:

- Distinct event of different processor can have same timestamp (partial ordering of events)
- Doesn't determine if the events are casually related or not.

These issues can be solved with the use of Vector clock.

Vector clock are the clock that provide the ability to determine if the two selected events are casually related or not. In vector clocks, each process maintains a vector of logical clock values, one for each process in the system. Vector clocks capture the "causal relationship" between events among different processes, providing a more accurate and consistent ordering of events. So, for N given processes, there will be vector/ array of size N.

Implementation:

- Each process P maintains a vector clock VC[P], where the i-th element VC[P][i] represents the logical clock value of process i at process P.
- When a process P initiates an event, it increments its own logical clock value: VC[P][P] += 1.
- When process P sends a message to another process Q, P includes its current vector clock VC[P] with the message.
- Upon receiving a message with vector clock VC', process Q updates its own vector clock element-wise:
    - For each i, VC[Q][i] = max(VC[Q][i], VC'[P][i]).

- Process Q then increments its own logical clock: VC[Q][Q] += 1.

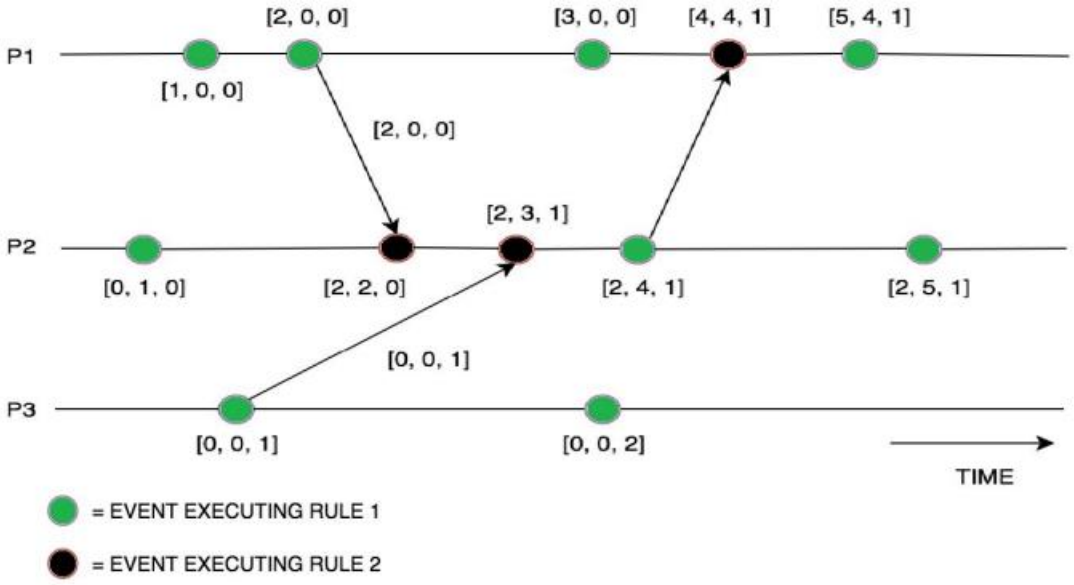


Fig: Example of Vector Clock Implementation

This approach ensures that the ordering of events respects the causal relationships between processes, regardless of the speed of their clocks. Vector clocks provide a more accurate and reliable way to track and compare the progress of events in a distributed system, making them suitable for various synchronization and consistency protocols.

Q.5 How does a new coordinator elect in executing central coordinator algorithm? How to come to consensus in DS? Explain.

Ans: Electing a New Coordinator in Central Coordinator Algorithm:

In distributed systems, the concept of a coordinator often arises in scenarios where multiple nodes need to coordinate their actions or make decisions collectively. The central coordinator algorithm is a common approach where one node is designated as the coordinator, and it manages the coordination process.

When it comes to electing a new coordinator in the central coordinator algorithm, the process typically involves the following steps:

1. Coordinator Failure Detection: The nodes in the distributed system continuously monitor the status of the current coordinator. If the coordinator

fails or becomes unresponsive, the other nodes detect this failure through various mechanisms such as heartbeat signals, timeouts, or communication failures.

2. Initiating Election: Once a node detects the coordinator's failure, it may initiate an election process to select a new coordinator. This node becomes a candidate for the coordinator role.
3. Sending Election Messages: The candidate node sends election messages to other nodes in the system, notifying them of its intention to become the new coordinator.
4. Priority Comparison: Upon receiving an election message, each node compares the priority of the candidate with its own priority or with the priorities of other candidate nodes. The priority can be based on factors like node ID, availability, load, or any other relevant criteria.
5. Choosing the New Coordinator: The node with the highest priority among all participating nodes becomes the new coordinator.
6. Announcing the New Coordinator: The elected node broadcasts an announcement to all nodes, indicating that it is the new coordinator. This ensures that all nodes are aware of the new coordinator's identity.

Coming to Consensus in Distributed Systems:

Consensus is the process of achieving agreement among multiple nodes in a distributed system on a specific value or decision. It is a fundamental challenge in distributed computing, especially when nodes may have different inputs or experiences. Achieving consensus is critical for ensuring the correctness and consistency of distributed applications. The process of coming to consensus typically involves the following steps:

1. Proposal Phase: Nodes propose a value or decision to be agreed upon. Each proposal is assigned a unique identifier.
2. Acceptance Phase: Nodes send their proposals to other nodes for acceptance. Nodes can accept multiple proposals but choose to accept the one with the highest identifier (highest priority).
3. Learning Phase: Once a proposal has been accepted by a majority of nodes, it is considered chosen. Nodes inform others about the chosen value.

4. Termination: The consensus process terminates when a value has been chosen by a majority of nodes and is agreed upon by all.

Q.6 What is fault? How to implement primary-backup replica system? How is it different from active replication?

Ans: A fault refers to an abnormal condition or behavior that disrupts the normal operation of a system, component, or process. Faults can manifest as errors, failures, or malfunctions and can occur due to various reasons, such as hardware or software issues, human errors, environmental factors, or external events.

The primary-backup replication model is a technique for achieving fault tolerance in distributed systems. It involves having a primary replica manager that handles client requests and one or more secondary replica managers (backups) that replicate the primary's state. If the primary replica manager fails, one of the backups is promoted to become the new primary. Steps to implement primary backup replica systems:

a. Designating Primary and Backups:
   - Choose one node to be the primary replica manager and one or more nodes to be secondary replica managers (backups).
   - The primary handles client requests and performs operations on the data.
b. Client Communication:
   - Clients send their requests to the primary replica manager.
   - Front-end communication is simplified as clients interact only with the primary.
c. Primary's Operation:
   - The primary executes received requests in the order they are received, maintaining a consistent view of the system's state.
   - It generates and assigns unique identifiers to each request to prevent re-execution of duplicates.
d. Updating Backups:
   - For update requests, the primary updates its own state and sends the updated data along with the response and unique identifier to all backup replica managers.
e. Backup Handling:

- Backup replica managers receive and store the updated state, response, and unique identifier.
- Backups remain passive and do not directly interact with clients.

f. Promotion on Primary Failure:
   - In the event of the primary's failure, a backup is promoted to become the new primary.
   - The promoted backup takes over client communication and data processing.

g. Client Response:
   - The primary replica manager responds to clients with the outcome of their requested operations.

It differs from active replication in following ways:

- Communication: In primary-backup replication, clients communicate only with the primary replica manager. In active replication, clients communicate with all replicas.
- Processing: In primary-backup, only the primary processes requests. In active replication, all replicas process requests independently.
- Complexity: Primary-backup is simpler to implement and manage since only one replica processes requests. Active replication involves more complexity due to the need to compare and reconcile responses.

Q.7 What do you mean by forward and backward recovery? How to implement coordinated check pointing for recovery in DS?

Ans:

Backward Recovery:

Backward recovery involves restoring a system from an incorrect or faulty state back to a previously accurate condition. The key challenge in backward recovery lies in effectively reversing the system's state to rectify errors. This is achieved through periodic recording of the system's state, creating checkpoints that mark accurate system states. In case of errors or failures, the system can be rolled back to a checkpoint, effectively undoing the effects of erroneous operations. Backward

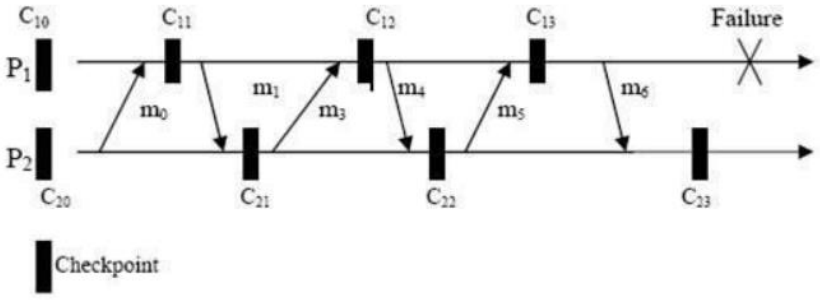
recovery is particularly useful when the system encounters unexpected issues, and it aims to bring the system back to a known and reliable state.

Forward Recovery:

Forward recovery aims to transition a system from its current state to a correct and operable new state. Unlike backward recovery, where errors are rectified by reversing to a known state, forward recovery focuses on predicting potential errors in advance. The main challenge here is identifying possible errors before they occur and taking proactive measures to prevent or mitigate their impact. Forward recovery techniques involve making changes to the system's operation to avoid known pitfalls or errors, ensuring the system can transition to a new, functional state while avoiding future errors.

Coordinated Check Pointing

Coordinated checkpointing is a technique used in distributed systems to achieve fault tolerance and recovery. It involves periodically saving the state of the entire distributed system so that, in the event of a failure, the system can be restored to a consistent state and continue its operation.



Coordinated checkpointing for recovery in a distributed system can be implemented through following steps:

1. Define Checkpoint Intervals: Determine how often you want to take checkpoints. This interval should strike a balance between minimizing the amount of lost work in case of failure and not introducing too much overhead due to frequent checkpointing.

2. Global Checkpoint Algorithm: Choose a coordinated checkpointing algorithm. One commonly used algorithm is the Chandy-Lamport Algorithm, which ensures that checkpoints are taken consistently across all processes in the system.
3. Process Coordination:
    - When a process decides to initiate a global checkpoint, it sends a marker message to all other processes to indicate that they should record their state.
    - Upon receiving a marker message, a process records its local state (including data, variables, and any necessary information for recovery) and sends an acknowledgment back to the initiator.
    - Once the process has received acknowledgments from all other processes, it records its own state and sends a message indicating that its state has been saved. This ensures that processes are aware of each other's checkpoint status.
4. Log-based Recovery:
    - In addition to checkpointing, maintain a log of all important events and actions taken by each process. This log can help replay events in case of failure to bring the system back to a consistent state.
    - Log-based recovery involves starting from the most recent consistent checkpoint and replaying events from the log to bring the system back to the correct state.
5. Failure Handling:
    - When a process detects a failure (e.g., a crashed process), it uses the recorded checkpoints and logs to determine the state at the time of failure.
    - The process then uses the recovery protocol to restore the system to a consistent state, potentially rolling back to a previous checkpoint and replaying events from the log.
6. Synchronization and Timing:
    - Ensure that the coordinated checkpointing algorithm takes into account network delays and message processing times to ensure that checkpoints are initiated and completed in the correct order.
7. Testing and Tuning:

- Implement and test the coordinated checkpointing mechanism thoroughly to ensure correctness and effectiveness in various failure scenarios.
- Tune parameters such as checkpoint intervals and log retention policies based on the characteristics of your distributed system.

8. Documentation and Monitoring:
   - Clearly document the coordinated checkpointing mechanism and the recovery process.
   - Implement monitoring and alerting to detect and respond to issues related to checkpointing and recovery.

Q.8 What are the alternative approaches to avoid the possibility of deadlock in distributed system? Explain.

Ans: Deadlocks in distributed systems can lead to resource contention and system inefficiencies. To avoid the possibility of deadlock, several alternative approaches and techniques can be employed. Here are some of them:

1. Resource Allocation Graphs:
   - Distributed systems can use resource allocation graphs to detect and prevent deadlocks. Nodes represent processes, and edges represent resource requests and allocations.
   - Techniques like Banker's algorithm can be extended to distributed systems, where the central authority monitors resource allocation to avoid unsafe states.
2. Timeouts and Rollbacks:
   - Introducing timeouts can help prevent deadlocks by allowing a process to release resources after a certain period of inactivity.
   - In distributed systems, if a process does not complete its operation within a specified time, the system can forcibly release its held resources, rolling back the operation.
3. Distributed Lock Management:

- Implement distributed lock management algorithms that ensure proper acquisition and release of locks to avoid deadlocks.
- Techniques like distributed lock managers (DLMs) can provide a centralized or distributed mechanism for coordinating locks among processes.

4. Priority-based Approaches:
   - Assigning priorities to processes can help prevent deadlocks by allowing higher-priority processes to preempt lower-priority processes.
   - In distributed systems, processes can communicate their priorities to a central authority that manages resource allocation.
5. Global States and Checkpointing:
   - Distributed systems can periodically capture global system states through checkpointing. In case of deadlock detection, the system can revert to a previous checkpoint state to avoid the deadlock.
6. Avoiding Circular Waits:
   - Ensure that processes request resources in a predefined order to avoid circular wait conditions.

Q.9 Write short notes on:

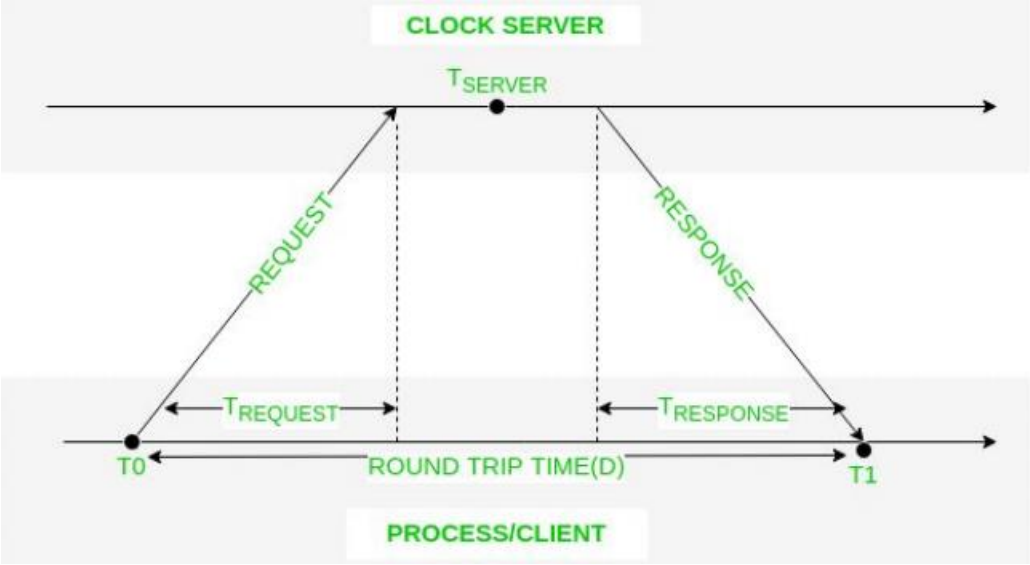
Physical Clock Synchronization: Cristain' Algorithm:

Cristian's Algorithm is a clock synchronization technique used to synchronize the time of client processes with a time server. It is particularly effective in low-latency networks where the Round-Trip Time (RTT) is short compared to the desired accuracy of time synchronization. Round Trip Time refers to the time duration between the start of a Request and the end of the corresponding Response. The steps involved in algorithm are:

I. The process on the client machine sends the request for fetching clock time(time at the server) to the Clock Server at time.
II. The Clock Server listens to the request made by the client process and returns the response in form of clock server time.
III. The client process fetches the response from the Clock Server at time and calculates the synchronized client clock time using the formula given below.
Tclient = Tserver + (T1 - To)/2

where,

Tclient = refers to the synchronized clock time,
Tserver = refers to the clock time returned by the server,
To = to the time at which request was sent by the client process,
T1 = refers to the time at which response was received by the client process



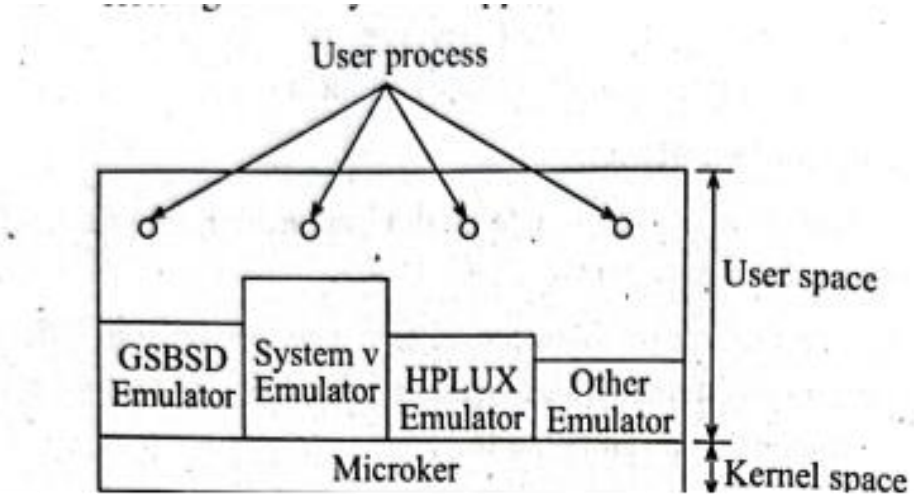
MACH

Mach, introduced in 1986, is a pioneering microkernel-based operating system. It offers the foundational services essential for an operating system to function, emphasizing modularity and extensibility. Originally developed for the VAX 11/784 multiprocessor, Mach quickly expanded its reach, becoming operational on a variety of systems including the IBM PC/RT and Sun B. Initially, Mach contained portions of Berkeley's code, leading to a relatively large and monolithic kernel. Subsequent efforts by CMU (Carnegie Mellon University) focused on purging Berkeley code from the kernel and relocating it to user space, aligning with the system's goals of supporting diverse architectures, functioning across varying inter-computer network speeds, simplifying the kernel's structure, and enabling distributed operations.

The goals of MACH are:

- Support diverse architectures

- Function with varying inter computer networks speeds
- Simplified kernel's structure
- Distributed operation
- Integrated memory management and IPC
- Heterogeneous system support
- Protected message passing
- Extensible microkernel



CORBA components for RMI

Common Object Request Broker Architecture (CORBA) is the standard developed by object management group to provide interoperability among the distributed objects. It describes the message mechanism by which object distributed over the network can communicate with each other irrespective of platform and language used to develop those objects. The components in CORBA are:

1. ORB core
   - It carries out the request-reply protocol between client and server.
   - It Provide operations that enable process to be started and stopped.
   - It Provide operations to convert between remote object references and strings.
2. Object Adapter (server)

- - Bridges the gap between CORBA objects and the programming language interfaces of the slave classes. Creates remoter object references for the CORBA objects
- Dispatches each RMI to the appropriate servant class via a skeleton, and activates objects.
- Assigns a unique name to itself and each object

3. Skeletons (server)
   - An IDL compiler generates skeleton classes in the server's language.
   - Dispatch RMI's to the appropriate servant class.
4. Client Proxies / Stubs
   - Generated by an IDL compiler in the client language.
   - A proxy class is created for object-oriented languages
   - Stub procedures are created for procedural languages.
5. Implementation Repository
   - Activates registered servers on demand and locates servers that are currently running.
6. Interface Repository
   - Provides information about registered IDL interfaces to the clients and servers that require it. Optional for static invocation; required for dynamic invocation.

Name: Ankit Bk. Ayush Badal, Nishan Thapa Ishwor Nishant Upadhy

2076 Ashoin (Back)

(Qno. 1)

1) what are the major goals of DS and challenges during design of D
→ the major goals of Distributed system are

i) Resource sharing:

The main goal of distributed system is to allow users to access remote resources and to share them in controlled and efficient manner. The resources may be printer, database, files etc. which is necessary for any collaboration and information exchange among the user

ii) openness:

A distributed system must be open. It must provide services with the standard rule following some protocols & defining syntax and service. The openess of system is pre-determined by the degree to which new resource can be added and be made available to users.

iii) Transparency:

Transparency is the ability to hide the fact that the process and resources are distribute across multiple network of computer, user to realize it as one coherent system. With the transparency the system is perieved as a whole than a collection of independent collection

(v) Scalability :

A distributed system is scalable if the cost of adding is a constant amount in terms of the resources that must be added. The algorithms used to access shared data should avoid performance bottlenecks and data should be structured hierarchially to get best access time.

The design challenges are

i) Transparency :

Complete Transparency is not always desirable due to the trade offs with performance and scalability as well as the problems can be caused while confusing local and remote operation

ii) Scalability :

In wider area, it is expected to grow largely. Growth in number of users and resources make a distributed system overloaded because of too many user requests

iii) Dependability

For dependability, all the system should work jointly only when the data are fragmented and distributed. Also, dependability also requires consistency, security, fault tolerance which are difficult to achieve.

1) Performance :

Any system should aim for maximum performance but incase of distributed system it directly conflicts with some other desirable properties like transparency, security, scalabity and easily be harmful to performance.

2) Define distributed object system and explain communication between distributed system

→ Distributed objects are the objects with respect to object oriented paradigm, that are distributed across different address spaces, which may be on multiple computers within a network or multiple processes running on a same computer. Distributed objects are capable to work together by storing and invoking methods.

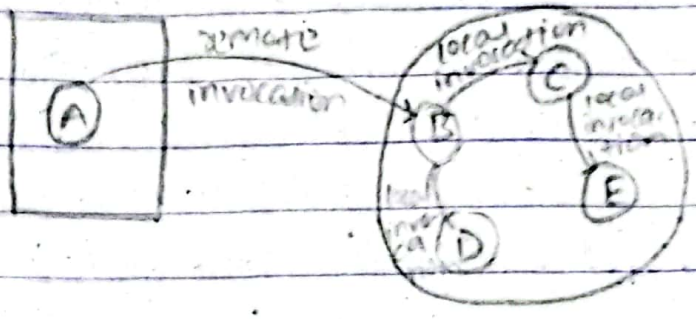
The communication between distributed system happens through two methods

1) Remote Method Invocation.

RMI is the means by which objects in different processes can communicate with one another. It allows to object in one process to invoke or call the methods of object in another process. Here object may receive local invocation or remote invocation or both.

The object which can receive remote invocation are called remote object. All the objects can receive local invocation from another object within a process if

and only if the invoking object have reference to invoked object

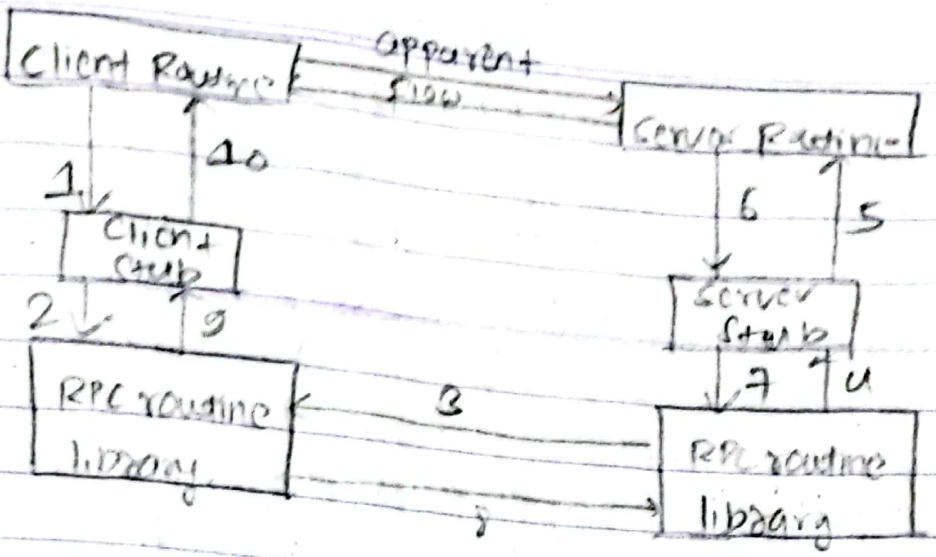


In the figure, object C can invoke object E locally because chas reference to E. Object A can invoke object B remotely so A must be have access to remote object reference of B

An action is invoked by a method invocation which may result in further invocations on methods in other objects and the object in this chain may be of different process. RMI should be able to raise exceptions if the remote invocation between distributed objects fails

ii) Remote procedure call :

It is a remote communication medium in which the a client program calls a procedure in another programming running in servers. The role of server and client designated based on the work which is temporarily assigned.



The working of RPC are

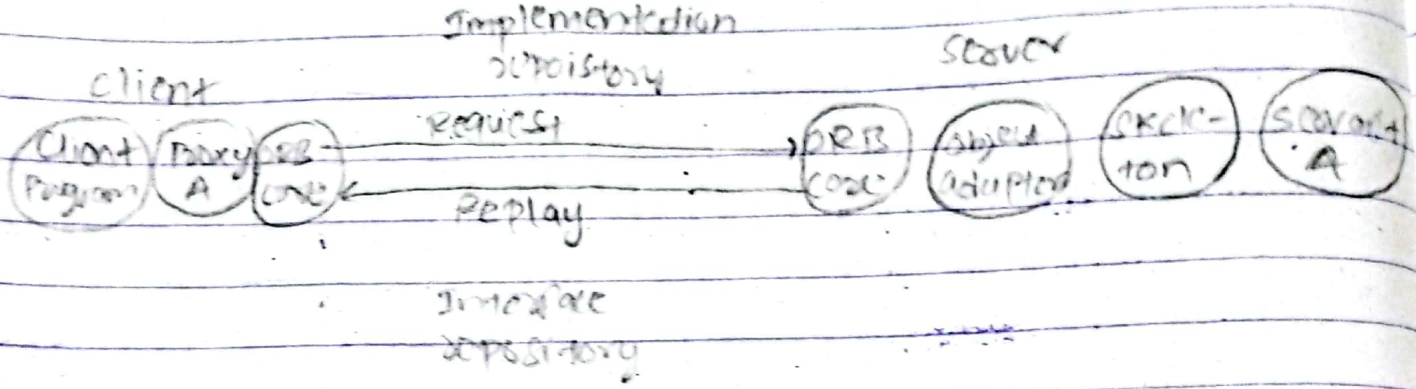
i) Client procedure calls client stub in normal way
ii) client stubs builds message, call local OS
iii) Client's OS sends message to remote OS.
iv) Remote OS gives message to server stub
v) server stubs unpacks parameter, call server
vi) server does work, return result to stub
vii) server stub packs it in message, call local OS
viii) Servers OS sends message to clients OS
ix) clients OS gives message to clients stub
x) stub unpacks result, returns to client

3) Define distributed file system? Point out the difference between a stateless and stateful services.

→ Distributed file system ~~enables~~ is a classical model of a file system distributed across multiple machine which enables programs to store and access remote file exactly as they do local ones access from any computer and leads to

Que 4) Explain components in CORBA architecture with a diagram

→



i) ORB core:

The role of ORB core is similar to that of the communication module. ORB core provide the ~~refer~~ interface which provides an interface that includes

i) ~~Enabling~~ enabling it to be started and stopped
ii) operations to convert between remote object reference and strings
iii) operation to provide argument list for request using dynamic invocation

ii) Object adapter (Server):

An object adapter has following tasks

i) It creates remote object reference for CORBA objects.
ii) It dispatches call via skeleton to appropriate servant
iii) It activates and deactivates servants

iii) Skeletons (server):

Skeleton classes are generated in the language of server by an IDL compiler. As before, remote method invocations are dispatched via skeleton to particular servant and the skeleton unmarshals the arguments in request message.

iv) client stubs/proxies:

These are in the client language and generated from IDL interfaces by an Interface compiler for client language. It marshal argument in invocation requests and unmarshals exceptions and result in replies.

v) Implementation repository:

It is responsible for activating registered servers on demand and for locating servers that are currently running.

vi) Interface repository:

The role of the Interface repository is to provide information about registered IDL interface to clients and servers that require it. It also can supply the names of the methods and for methods, the names and types of arguments and exceptions.

ans.5)

Explain lamport logical clock with its proc and cons.

→ To synchronize ~~lam~~logical clock, lamport defined a relation happen before. The expression $a \rightarrow b$ read as "event a happens before event b" and means all process agree that Event A has occured before Event B. This happens before relation can be observed into two relations

i) If a and b are events in same process then $a \rightarrow b$ is true

ii) If A is the event of message being sent by one process and b is the event of message being received by another process then $a \Rightarrow b$ is true

The ordering of the event also can be observed in two situations

1.) If two events within a process occured, they are occured in the order in which process sees it

2) when ever message is sent between processes the event of sending message occured before the event of receiving message.

Implementation

i) L_i is incremented before each event is issued at process P_i

ii) with send (m) is a event of P_i

P_i restamp $t_m = L_i$ is included in m's

iii) on recieve (m) by P_j its clock is updated $L_j = max(L_j, t_m)$. The new value of L_j is used to timestamp

ans. 5)

Explain lamports logical clock with its pros and cons

→ To synchronize lamlogical clock, lamport defined a relation happen before. the expression a→b read as "event a happens before event b" and means all process agree that Event A has occured before Event B. This happens before relation can be observed into two relations

i) If a and b are events in same process then a→b is true

ii) If a is the event of message being send by one process and b is the event of message being received by another process then a→b is true

The ordering of the event also can be observed in two situations

1.) If two events within a process occured, they are occured in the order in which process sees it

2) when ever message is send between processes the event of sending message occured before the event of receiving message.

Implementation

i) Li is incremented before each event is executed process Pi

ii) with send (m) is a event of Pi

Pi receives tm: Li is included in m

iii) on recieve (m) by Pj its clock is updated Lj: max (Lj, tm). The new value of Lj is used to timestamp

event P_j recieve (m) by P_j

Advantage

i) lamports clock are easy to understand and implement.
ii) It detects causual relationship between events.
iii) It does not require a global clock

disadvantage

i) Distinct event can have same time stamp in diff process
ii) It does not provide real time ordering of system

(Qno.6)

→ Explain reliable multicast with its properties and an algorithm

Ans Reliable multicast is a communication pattern in distributed systems where a message is sent from one sender to multiple receivers, ensuring that all the intended recipients successfully recieve the message and in correct order.

The properties of reliable multicast are

i) message are delivered to all the receiver, in same order
ii) All correct processes that are supposed to receive will recieve the message which makes the reliable
iii) messages are not duplicated
iv) This protocol works correctly even in the presence of network failures

The algorith of the Reliable FIFO multicast with acknowledge algorithm are.

Sender steps
1) Sender maintains a sequence number for each message.
2) when sending a message, increment sequence number and attach to message
3) Send message to all the receivers
4) wait for acknowledgements from all receivers. If acknowledgement not received, resend message

Receiver step
1) Receiver maintains an expected sequence number.
2) when receiving a message, deliver to process and increment expected
3) If out of order, buffer the message.
4) Send acknowledgement to sender

(ans.7)

Specify Data centric consistency models and explain any one of them in detail?

→ The data centric consistency models are
1) strict consistency
2) sequential consistency
3) casual consistency
4) Processor consistency
5) Release consistency
6) Entry consistency.

strict consistency:

This consistency makes use of absolute time ordering of all shared resources. It is unrealistic for distributed system as it requires absolute global time. A data store is said to be strict consistent when it satisfies the following condition i.e. any read to shared data should return value stored by the most recent write operations.

eg:

Consider a system with two processes P1 and P2. let P1 initiate an write operation for variable x with value a. the model is strictly consistent if and only if, the read operation for variable x from both P1 and P2 just after the completion of write operation return a.

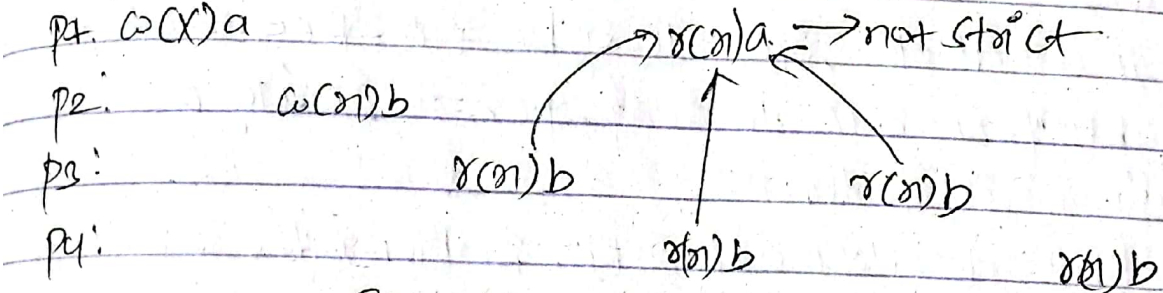


Fig: strict consistency.

(ans. 8)

Why is it necessary to maintain transaction? what is a deadlock and what are phantom deadlock

→ It is necessary to maintain transaction because of the following.

1) Transactions ensure that the data remains in a valid and consistent state. preventing incorrect data modifications
2) Maintaining transactions ensures the atomicity that a transactions complete successfully or not
3) Transactions ensures predefined rule, constraints ensuring the database remains consistent
4) Transactions provide isolation between concurrent operations which prevents interference, data conflicts etc.

Deadlock occurs in concurrent systems when two or more processes are unable to proceed because each process is waiting for a resource that other holds. Deadlock can cause system stalls and performance issues

Phantom deadlocks are situations where the system incorrectly identifies a deadlock, even though one doesn't actually exist. This can happen due to improper or overly aggressive deadlock detection algorithms.

Phantom deadlocks can be problematic because they can lead to unnecessary resource blocking or process terminations impacting system performance and causing delays. Detecting actual deadlocks is essential for effective resource management in distributed system.

(ans. 9)

what is fault tolerance? Explain different types of fault that may occur in a distributed system.

→ A system is said to be fault tolerance if it can continue or ~~~~ function in presence of failures.
Types of fault that may occur in a distributed system are.

i) Node fault:
A fault that occured at an individual node participating in the distributed system is called node fault.
eg: A machine in distributed system fails due to some error in database configuration

ii) Program fault
A fault that occured due to some logical or syntactical error in the code called program fault
eg: A program that is designed filter the data by category but instead filtering by other means

iii) Communication Fault
A fault that occured due to unreliable communication connecting to the node.
eg: A node sending data "01010" to another node but due to some problem in communication channel the reciever recieves "01001"

d) ~~Fault~~ Timing Fault.

The fault that occured due to mismatch or timing of any particular response is called Timing Fault.

eg: A server replies too late or server is provided with data too soon that has no longer clear buffer to hold the data.

(no. 10)

b) Monolithic and micro Kernel

Monolithic Kernel:

Monolithic Kernel is a single large process address space. All Kernel services execute in the Kernel address space. It provides faster execution as there is no neccessary of address space switching to execute Kernel services the device driver resides in Kernel space making it less secure. fault in a single Kernel space collapse the whole Kernel

eg: Kernel in UNIX and LINUX

micro Kernel:

micro kernel is a kernel in which it is broken into separate process called servers. servers run in Kernel space and user space. It provides slow execution. fault in single server doesnot collapse the kernel

eg: Kernel of Mac OS X & windows NT.

Q MACH:

MACH is a pioneering microkernel based operating system which played a vital role in the field of operating system and separates operating system into small, essential core and various user level processes that provide additional functionality.

Mach emphasizes efficient and flexible interprocess communication mechanisms. Processes communicate through messages, enabling data and control message to be passed between different address space. Mach also provides virtual memory management which is a critical component for providing advanced memory features such as demand paging, memory isolation. It also provides port based communication, Task and thread management and so on.

Mach's design principle have had a long impact on the development of operating systems and distributed system.