TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

An Assignment Of Distributed Systems On 2070 Regular

Submitted By:

Amit Raj Pant THA076BCT005

Bishal Rijal THA076BCT014

Kshitiz Poudel THA076BCT018

Pilot Khadka THA076BCT025

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

August, 2023

2070 Chaitra (Regular)

1. Define Distributed System, what are advantages and disadvantages of distributed system?

➔ A distributed system is a collection of independent computers that appears to its users are a single coherent system. It is one in which components are distributed across multiple networked computers and can communicate and coordinate actions by sending messages to one another.

Advantages of a Distributed System

i. The price/performance ratio of a distributed system is higher(a cost-effective way to increase computing power).

ii. Each components in distributed system could work independently to solve problems, hence components are efficient on distributed system.

iii. More nodes can be easily added to the distributed system, allowing it to scale as needed.

iv. Failure in a single node does not result in the failure of the entire distributed system.

v. The distributed system's nodes are all linked together which allow nodes to share data with one another.

Disadvantages of Distributed System

i. In comparison to a single-user system, the startup cost of a distributed system is higher.

ii. It is difficult to provide adequate security in distributed systems because both nodes and connections must be protected.

iii. In comparison to a single-user system, the database connected to distributed systems in complicated and difficult to manage.

iv. Overhead is a common problem faced by a distributed system.

v. Distributed systems are prone to network errors which results in communication breakdown

2. Draw and explain distributed file service architecture. How does that architecture encourage the sharing of storage resources in distributed system? Explain.

➔ The file service is structured as three components – a flat file service, a directory service, and a client module – to provide a clear separation of the main concerns in providing access to files. Each of the flat file and directory services exports an interface for use by client programs, and their RPC interfaces when combined provide a comprehensive set of operations for file access. The client module offers a single programming interface for file operations similar to those found in traditional file systems. Different client modules can be used to implement different programming interfaces, simulating file operations of various operating systems and optimizing performance for different client and server hardware configurations.

The division of responsibilities between the modules can be defined as follows:

1. Flat file service
2. Directory file service
3. Client module

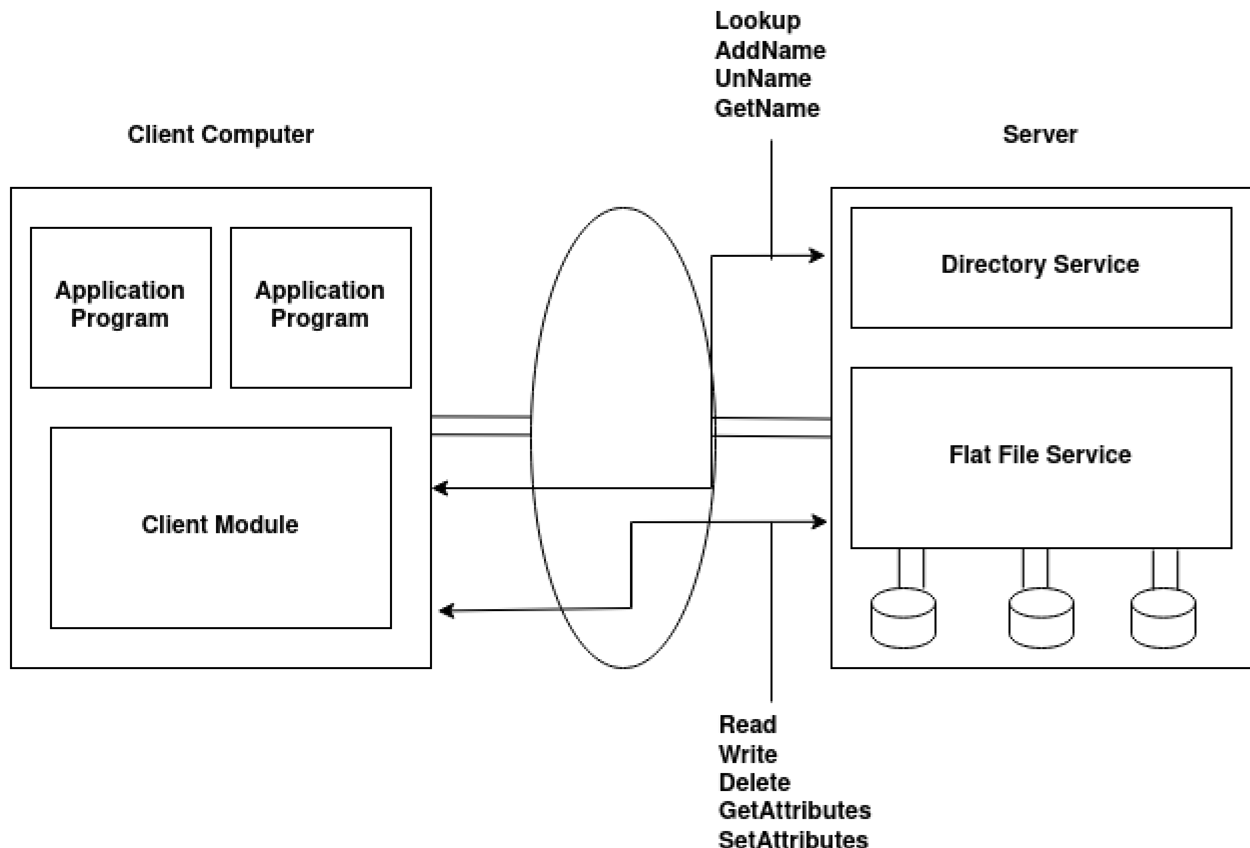


Fig: File Service Architecture

1. Flat File Service

   The *flat file service* is responsible for performing operations on file contents. In all requests for flat-file service operations, unique file identifiers (UFIDs) is used to refer to files. The use of UFIDs is used to divide responsibilities between the file service and the directory service. UFIDs are long sequences of bits that are chosen so that each file in a distributed system has a UFID that is unique. When a request to create a file is received, the flat file service generates a new UFID for it and returns it to the requester.

2. Directory File Service

   The *directory file service* maintains a mapping between file text names and UFIDs. Clients can get the UFID of a file by giving the directory service its text name. The directory service includes functions for creating directories, adding new file names to them, and retrieving UFIDs from them. It is a flat-file service client, and its directory files are stored in flat file service files. Directories hold references to other directories when a hierarchical file-naming scheme is used, such as in UNIX.

3. Client Module

   A *client module* runs in each client computer, integrating and extending the operations of the flat file service and the directory service under a single application programming interface that is available to user-level programs in client computers. For example, in UNIX hosts, a client module would be provided that emulates the full set of UNIX file operations, interpreting UNIX multi-part file names by iterative requests to the directory service. The client module also holds information about the network locations of the flat file server and directory server processes. Finally, the client module can play an important role in achieving satisfactory performance through the implementation of a cache of recently used file blocks at the client.

   The distributed file service architecture encourages the sharing of storage resources in a distributed system by allowing multiple clients to access the same files. This is done by replicating the files across multiple storage servers. This replication ensures that the files are always available, even if one of the storage servers fails. In addition, the distributed file service architecture allows for load balancing. This means that the files can be distributed across multiple storage servers so that no single server is overloaded. This can improve performance and reliability.

4. Differentiate between RPC and RMI. How does modern RPC maintain the transparency in distributed system?

	RPC	RMI
1.	RPC is a library and OS dependent platform.	Whereas it is a java platform.
2.	RPC supports procedural programming.	RMI supports object-oriented programming.
3.	RPC is less efficient in comparison of RMI.	While RMI is more efficient than RPC.
4.	RPC creates more overhead.	While it creates less overhead than RPC.
5.	The parameters which are passed in RPC are ordinary or normal data.	While in RMI, objects are passed as parameter.
6.	RPC is the older version of RMI.	While it is the successor version of RPC.
7.	There is high Provision of ease of programming in RPC.	While there is low Provision of ease of programming in RMI.
8.	RPC does not provide any security.	While it provides client level security.
9.	It's development cost is huge.	While it's development cost is fair or reasonable.
10.	Can be complex due to low-level implementation	Generally simpler to use and implement

Modern RPC maintains transparency in distributed systems by using a variety of techniques, including:

- **Stubs and skeletons**: Stubs and skeletons are objects that are used to encapsulate the details of the remote procedure call. The stub is used on the client side, and the skeleton is used on the server side. The stub and the skeleton take care of the details of the communication between the client and the server, such as serialization and deserialization of data, and error handling. This allows the client and the server to communicate with each other without having to know the details of the underlying network protocol.

- **Call backs**: Call backs are used to notify the client when the remote procedure call has completed. This allows the client to continue executing its code without having to wait for the response from the server.

- **Timeouts**: Timeouts are used to ensure that the remote procedure call does not block the client for too long. If the remote procedure call does not complete within the specified timeout, the client will be notified and can take appropriate action.

- **Error handling**: Modern RPC frameworks provide a variety of mechanisms for handling errors that occur during a remote procedure call. This allows the client to recover from errors gracefully and continue executing its code.

By using these techniques, modern RPC frameworks can maintain transparency in distributed systems and make it easy for programmers to develop distributed applications. In addition to the techniques mentioned above, modern RPC frameworks also often support other features that can help to improve the performance and reliability of remote procedure calls, such as:

- **Caching**: Caching can be used to store the results of remote procedure calls in memory so that they do not have to be recomputed every time the call is made. This can improve performance by reducing the amount of network traffic and the amount of computation that needs to be performed.

- **Load balancing**: Load balancing can be used to distribute remote procedure calls across multiple servers so that no single server is overloaded. This can improve performance and reliability by preventing any one server from becoming a bottleneck.

- **Security**: Modern RPC frameworks often support a variety of security features, such as authentication, authorization, and encryption, to protect remote procedure calls from unauthorized access and modification.

4. Compare process and threads. Why threads are important in distributed System?

S. No	Process	Thread
1.	Process means any program is in execution.	Thread means a segment of a process.
2.	The process takes more time to terminate.	The thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	The process is less efficient in terms of communication.	Thread is more efficient in terms of communication.
6.	Multiprogramming holds the concepts of multi-process.	We don’t need multi programs in action for multiple threads because a single process consists of multiple threads.
7.	The process is isolated.	Threads share memory.
8.	The process is called the heavyweight process.	A Thread is lightweight as each thread in a process shares code, data, and resources.

Threads are important in distributed systems for a number of reasons, including:

- **Improved performance**: Threads can improve the performance of distributed systems by allowing multiple tasks to be executed concurrently. This is because threads share the same address space and resources, so they do not need to incur the overhead of creating and destroying processes.

- **Increased scalability**: Threads can help to increase the scalability of distributed systems by allowing them to handle more requests without having to increase the number of processes. This is because threads can be created and destroyed more easily than processes, so they can be scaled up or down as needed.

- **Better resource utilization**: Threads can help to improve the resource utilization of distributed systems by allowing multiple tasks to share the same resources. This can help to reduce the overall cost of running a distributed system.

- **Simplified programming**: Threads can make it easier to program distributed systems by allowing developers to break down complex tasks into smaller, more manageable units. This can make it easier to debug and maintain distributed systems.

5. Give an example of heterogeneous model of distributed application. How is distributed operating system realized in practical distributed systems? Explain.

➔ A heterogeneous model of distributed application is one in which the computers that make up the distributed system have different hardware and software platforms. This can make it more challenging to develop and deploy applications on a heterogeneous distributed system, but it can also offer some advantages.

One example of a heterogeneous distributed application is a scientific computing application that uses a cluster of computers to run a complex simulation. The computers in the cluster may have different processors, memory, and operating systems, but they can all be used to contribute to the simulation. This can allow the simulation to be run on a larger scale than would be possible on a single computer.

Another example of a heterogeneous distributed application is a web application that is hosted on a cloud computing platform. The cloud computing platform may use a variety of different servers to host the application, and each server may have its own operating system and hardware configuration. This can make it more challenging to develop and deploy the web application, but it can also offer some advantages, such as scalability and cost-effectiveness.

A distributed operating system (DOS) is a software system that manages a distributed system. It provides the services that are necessary for the computers in the distributed system to communicate with each other and to share resources.

There are two main approaches to realizing a DOS in practical distributed systems:

- **Centralized DOS:** In a centralized DOS, there is a single computer that acts as the central authority for the distributed system. This computer is responsible for managing all of the resources in the system and for coordinating the activities of the other computers.
- **Decentralized DOS:** In a decentralized DOS, there is no single central authority. Instead, the computers in the distributed system cooperate to manage resources and to coordinate their activities.

The centralized DOS approach is simpler to implement, but it can be less scalable and more vulnerable to failures. The decentralized DOS approach is more complex to implement, but it can be more scalable and more resilient to failures.

The choice of which approach to use depends on the specific requirements of the distributed system. For example, a centralized DOS may be a good choice for a small distributed system with a limited number of computers. A decentralized DOS may be a good choice for a large distributed system with a large number of computers.

6. What do you mean physical and logical clocks? Explain Network Time Protocol and Berkeley Algorithm for physical clock synchronization.

➔ In distributed systems, each computer has its own clock, which is called a physical clock. Physical clocks are not perfectly synchronized, so they can drift apart over time.

A logical clock is a mechanism for assigning a logical timestamp to events in a distributed system. Logical timestamps are used to order events and to establish causality between them. This allows distributed systems to deal with the fact that physical clocks are not perfectly synchronized.

Network Time Protocol (NTP)

Cristian's method and the the Berkeley algorithm were created with Intranets in mind. Mills (1995) defined the Network Time Protocol (NTP) as an architecture for a time service and a protocol for distributing time data over the Internet. The synchronization of hosts on a TCP/IP network is handled by this protocol, which is an application protocol.

NTP is an architecture that allows clients across the internet to be accurately synchronized to a universal time cordinate (UTC). It synchronizes with multiple time servers.

The design aims of NTP are:

- To provide a service enabling clients across the Internet to be synchronized accurately to UTC.
- NTP servers have access to atomic clocks and GPU clocks that are extremely accurate.
- To provide a reliable service that can survive lengthy losses of connectivity.
- To enable clients to re-synchronize sufficiently frequently to offset the rates of drift found in most computers.
- To provide protection against interference with the time service, whether malicious or accidental.

Advantages of NTP:

- It synchronizes the devices' Internet connections.
- It increases the level of security within the building.
- It's used in Kerberos and other authentication systems.
- It helps troubleshoot problems by providing network acceleration.
- Used in file systems where network synchronization is difficult.

Disadvantages of NTP:

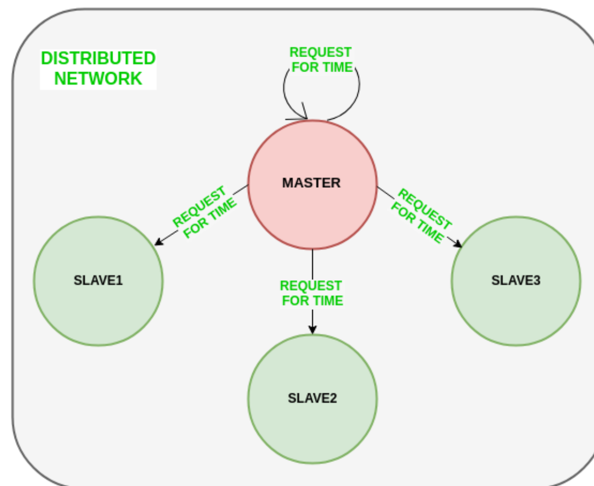
- The sync time of a running communication is affected when the servers are down.
- When the number of NTP packets is increased, synchronization is conflicted.
- Due to the various time zones, servers are prone to errors, and conflicts may arise.

Applications of NTP:

- Used in a production system that records live sound.
- Broadcasting infrastructures are being developed.

Berkeley’s Algorithm

The Berkeley algorithm uses internal synchronization. Unlike Cristian’s protocol, this computer polls the slaves, or computers whose clocks must be synchronized, on a regular basis. It receives the clocks must be synchronized, on a regular basis. It receives the clock values from the slaves. The master calculates their local clock times by observing round-trip times (a similar to Cristian’s) and averaging the results (including its own clock’s reading). On balance of probabilities, this average cancels out the individual clocks’ fast or slow running tendencies. The protocol’s accuracy is determined by a nominal maximum round-trip time between the master and slaves. Any readings associated with longer times than this maximum are eliminated by the master.



Algorithm:

1. A master is chosen via an election process.

2. The master polls the slaves who reply with their time in a similar way to Christian’s algorithm.

3. The master observes the round trip time (RTT) of the messages and estimate the time of each slave & it’s own.

4. The master then averages the clock times, ignoring any values it receives for outsides the value of the others.

5. Instead of sending the updated current time back to the process, the master then spends out the amount (positive or negative) that each slave must adjust its clock.

Drawbacks:

i. Time is not a reliable method of synchronization.

ii. Users mess up the clock (and forget to set their time zones).

iii. Unpredictable delay on the Internet.

iv. Relativistic issues

If A and B are far apart physically and two events T_A and T_B are very close in time, then which comes first? There is a confusion.

7. How does mutual exclusion help in co-ordination in distributed system? Explain the way how Lamport algorithm ensures mutual exclusion?

Mutual exclusion is a fundamental concept in distributed systems that ensures that only one process or thread can access a shared resource at a time. This helps in coordinating the activities of multiple processes to avoid conflicts and maintain data consistency. In a distributed system, where processes are running on different machines and communicating over a network, achieving mutual exclusion becomes more challenging due to potential communication delays, failures, and lack of a centralized clock. The Lamport algorithm is one approach to achieving mutual exclusion in such distributed systems.

Lamport's Algorithm for Mutual Exclusion:

The Lamport algorithm is based on logical timestamps (Lamport timestamps) assigned to events in a distributed system. These timestamps are used to order events across different processes, allowing the system to maintain a consistent view of the sequence of events.

Requesting Critical Section:

When a process wants to enter a critical section (access a shared resource), it follows these steps:

a. The process sends a REQUEST message to all other processes, the request message will contain its own Lamport timestamp. The timestamp reflects the order of the requesting process's events.

b. The process waits for replies from all other processes. It only enters the critical section once it receives a REPLY message from all other processes and their timestamps are greater than its own.

Granting Permission:

When a process receives a REQUEST message from another process, it follows these steps:

a. If the receiving process is not currently in its critical section, it replies with a REPLY message and updates its own timestamp to be greater than the requesting process's timestamp.

b. If the receiving process is currently in its critical section, it defers the reply until it exits the critical section. This ensures that no two processes are in the critical section simultaneously.

Exiting Critical Section:

After a process finishes its critical section, it sends RELEASE messages to all other processes, indicating that it has left the critical section. This step allows other processes waiting to enter the critical section to proceed.

The Lamport algorithm guarantees that processes will enter the critical section in the order defined by Lamport timestamps. However, it may allow some unnecessary delays and potentially cause starvation if a higher timestamp process gets stuck in a loop of requesting access.

8. What are the major objectives for replication in distributed system? Explain primary backup model for fault tolerance?

Replication in distributed systems involves creating and maintaining multiple copies of data or services across different nodes or servers. The primary objectives of replication in distributed systems include:

1. Fault Tolerance and High Availability: Replication enhances the system's fault tolerance by ensuring that if one replica (copy) of data or service becomes unavailable due to a hardware failure, network issue, or other reasons, the system can continue to operate using the available replicas. This improves system availability and reduces downtime.

2. Improved Performance and Scalability: Replication can lead to improved performance by distributing the load across multiple replicas. Clients can be directed to the nearest or least loaded replica, reducing response times and increasing the system's ability to handle a higher number of requests.

3. Reduced Latency: By placing replicas closer to the users or clients, replication can help reduce network latency and improve response times for accessing data or services.

4. Load Balancing: Replication allows for load balancing, where requests are distributed among replicas to ensure even distribution of workload and prevent any single replica from being overwhelmed.

5. Concurrency and Parallelism: Replication can enable concurrent access to data, allowing multiple users or processes to read and update the same data simultaneously.

One common approach to achieving fault tolerance through replication is the Primary-Backup Model. In this model, there is a primary replica (also known as the master) and one or more backup replicas (also known as slaves). The primary replica handles all read and write operations, while the backup replicas remain passive and do not respond to client requests.

Here's how the Primary-Backup Model works:

1. Normal Operation:

- All client requests are directed to the primary replica.

- The primary replica processes read and write requests, updates its state, and sends updates to the backup replicas.

2. Fault Detection:

- If the primary replica fails or becomes unreachable, a fault detection mechanism detects the failure.

3. Promotion of Backup Replica:

- One of the backup replicas is promoted to become the new primary replica. This process is known as failover.

4. Continued Operation:

- The newly promoted primary replica takes over the responsibilities of handling client requests.

- The failed primary replica, once restored, becomes a backup replica and starts receiving updates from the new primary.

Benefits of the Primary-Backup Model:

- Simple Failover: The failover process is relatively straightforward, as only one backup replica needs to be promoted to become the primary replica.

- Data Consistency: The primary replica ensures that updates are properly synchronized with backup replicas, maintaining data consistency.

- Isolation of Failures: Failures are isolated to individual replicas, minimizing the impact on the overall system.

9. Differentiate between nested transaction and distributed transaction with examples. How is commitment ensured in distributed transaction?

Nested transactions refer to a situation where a transaction contains one or more sub-transactions, forming a hierarchical structure. Each sub-transaction is treated as an individual unit of work, and the success or failure of a sub-transaction affects the overall outcome of the parent transaction.

For example, consider a banking system where a user wants to transfer money from one account to another. The entire operation involves two sub-transactions: withdrawing money from the source account and depositing it into the destination account. If either of these sub-transactions fails, the entire transfer is rolled back, ensuring data consistency.

Distributed Transactions:

Distributed transactions involve multiple independent databases or resources located on different machines or nodes. A distributed transaction spans across these multiple resources and ensures that they are all updated consistently, as if they were part of a single transaction.

For example, imagine an e-commerce application where a customer places an order. The transaction involves updating the order information in the orders database, reducing the product quantity in the inventory database, and charging the customer's credit card through a payment gateway. All of these updates need to be performed atomically, ensuring that if any part of the transaction fails, the entire operation is rolled back to maintain data integrity.

Ensuring Commitment in Distributed Transactions:

Ensuring commitment (or atomicity) in distributed transactions is a crucial aspect to maintain data consistency across multiple resources. Commitment is the process of making sure that all changes made within a distributed transaction are either applied to all resources involved or none at all. There are two main protocols used to ensure commitment in distributed transactions: Two-Phase Commit (2PC) and Three-Phase Commit (3PC).

Two-Phase Commit (2PC):

1. Preparation Phase: In this phase, the coordinator (typically the client or an orchestrating service) sends a prepare request to all participants (resources involved in the transaction) to confirm their readiness to commit. Participants respond with a "yes" or "no" vote.

2. Commit Phase: If all participants vote "yes," the coordinator sends a commit message to all participants. Upon receiving the commit message, each participant applies the changes and confirms the commit. If any participant votes "no," the coordinator sends an abort message to all participants, and the transaction is rolled back.

Three-Phase Commit (3PC):

Three-Phase Commit is an enhancement to 2PC that introduces an extra phase to handle situations where participants can't be certain of their decision due to communication failures.

1. CanCommit Phase: Similar to the preparation phase in 2PC, participants send a "can commit" message to the coordinator. Instead of a simple "yes" or "no," participants can respond with "commit," "abort," or "wait."

2. PreCommit Phase: The coordinator collects the responses and decides whether to commit, abort, or wait. If the decision is to commit, participants prepare for commit. If the decision is to abort, participants release any resources they have acquired. If the decision is to wait, participants await further instructions.

3. DoCommit Phase: The coordinator sends a final decision to participants. If the decision is to commit, participants proceed with the commit. If the decision is to abort, participants roll back the transaction. If the decision is to wait, participants continue to wait.

10. What do you mean by fault tolerant system? What do you mean by Byzantine Failure? Explain Byzantine Generals problem to illustrate how agreement can be reached in faulty system?

A fault-tolerant system is a type of computer system or network that is designed to continue functioning and providing its services even in the presence of hardware failures, software errors, or other unexpected disruptions. The goal of a fault-tolerant system is to ensure that the system can maintain its critical operations and data integrity despite the occurrence of failures.

Byzantine Failure: Byzantine Failure refers to a scenario in which components or nodes in a distributed system exhibit arbitrary, unpredictable, and potentially malicious behavior. This type of failure is particularly challenging because the faulty components may send contradictory or incorrect information to other parts of the system, leading to confusion and incorrect decision-making.

Byzantine Generals Problem: The Byzantine Generals Problem is a famous thought experiment in distributed computing that illustrates the challenges of achieving consensus in a faulty or unreliable communication system. In this problem, a group of Byzantine generals, each commanding a portion of an army, need to agree on a common plan of action—whether to attack or retreat. The generals can only communicate by sending messages, but some of the generals (up to one-third of them) may be traitors and send conflicting messages to confuse the loyal generals.

The challenge is to devise a protocol that allows the loyal generals to reach a consensus despite the presence of traitors. The solution must satisfy the following conditions:

1. Agreement: All loyal generals agree on the same plan of action (attack or retreat).

2. Validity: The chosen plan of action must be one of the options proposed by a loyal general.

3. Termination: The protocol must eventually reach a decision.

A naive approach, such as a simple majority vote, may not work in the presence of Byzantine failures. Traitor generals could sway the vote by sending contradictory messages to different generals, preventing them from reaching a consistent decision.

To solve the Byzantine Generals Problem, the algorithm needs to be robust against a certain number of traitors. One solution is the "Practical Byzantine Fault Tolerance" (PBFT) algorithm, which is designed to tolerate up to one-third of faulty or malicious participants. In PBFT:

1. A commander (leader) is elected to initiate the consensus process.

2. The commander sends a proposal to the other generals.

3. Generals exchange messages to validate the proposal and ensure that a supermajority (two-thirds) agrees on the same plan of action.

4. If a supermajority agrees, the generals commit to the plan and broadcast their commitment.

5. Once enough commitments are received, the generals know that the consensus has been reached and can proceed with the agreed-upon action.

PBFT guarantees that the loyal generals will reach consensus even if up to one-third of the generals are traitors. It uses a multi-round process with cryptographic signatures and voting to ensure that the generals agree on a plan of action despite Byzantine failures and malicious behavior.

11. Write short notes on: [3+3]

a) Comparison of CORBA and Mach

b) Timestamp ordering in concurrency control

a) CORBA:

- CORBA is a middleware technology that facilitates communication and interaction between distributed objects across different platforms and programming languages.
- It enables interoperability by defining a standard interface description language (IDL) that specifies the interfaces of distributed objects, allowing clients and servers to communicate seamlessly.
- CORBA provides a request broker, responsible for managing communication between clients and servers, handling object references, and invoking remote methods.
- It is primarily used for building distributed applications, including those involving various software components, services, and systems.

Mach:

- Mach is an operating system kernel developed at Carnegie Mellon University. It is designed to support the development of distributed systems and provides features like process management, virtual memory, and inter-process communication (IPC).
- Mach's IPC mechanisms include message passing and remote procedure call (RPC) facilities, allowing processes to communicate across different machines.
- It offers a microkernel architecture, where core operating system services are separated into different user-level and kernel-level tasks.
- Mach's focus is on providing a flexible and modular foundation for building distributed and parallel computing environments.

b) Timestamp Ordering in Concurrency Control:

Timestamp ordering is a technique used in concurrency control mechanisms to enforce serializability and ensure correct execution of transactions in a multi-user database system. It assigns a unique timestamp to each transaction or operation based on the order in which they are executed. The timestamps are used to determine the order in which transactions are allowed to access and modify data, preventing conflicts and ensuring data consistency.

Key concepts of timestamp ordering include:

- Transaction Timestamps: Each transaction is assigned a timestamp that reflects its order of initiation or start time.

- Transaction Serialization: Transactions are serialized or ordered based on their timestamps. A transaction with an earlier timestamp is allowed to execute before a transaction with a later timestamp.

- Conflict Resolution: If two transactions attempt to access the same data concurrently, the system uses their timestamps to determine which transaction should proceed. The transaction with the earlier timestamp is typically given priority, while the other transaction might be aborted or delayed.

- Transaction Commit: Once a transaction completes its execution, it can be committed and its changes are made permanent in the database.

- Ensuring Serializability: Timestamp ordering ensures that the execution of transactions maintains serializability, meaning that the final state of the database is equivalent to some serial order of executing the transactions.

For example, consider two transactions T1 and T2. If T1 starts before T2, it will be assigned an earlier timestamp. If T2 attempts to read or modify data that T1 has already accessed, the timestamp ordering mechanism will ensure that T2 waits until T1 completes, preserving the correct order of execution.

While timestamp ordering is effective in ensuring serializability and preventing conflicts, it might lead to situations where transactions are unnecessarily delayed (e.g., due to contention for resources). More advanced concurrency control techniques, such as optimistic concurrency control and multi-version concurrency control, address some of these limitations and offer improved performance in high-concurrency scenarios.