Assignment 1

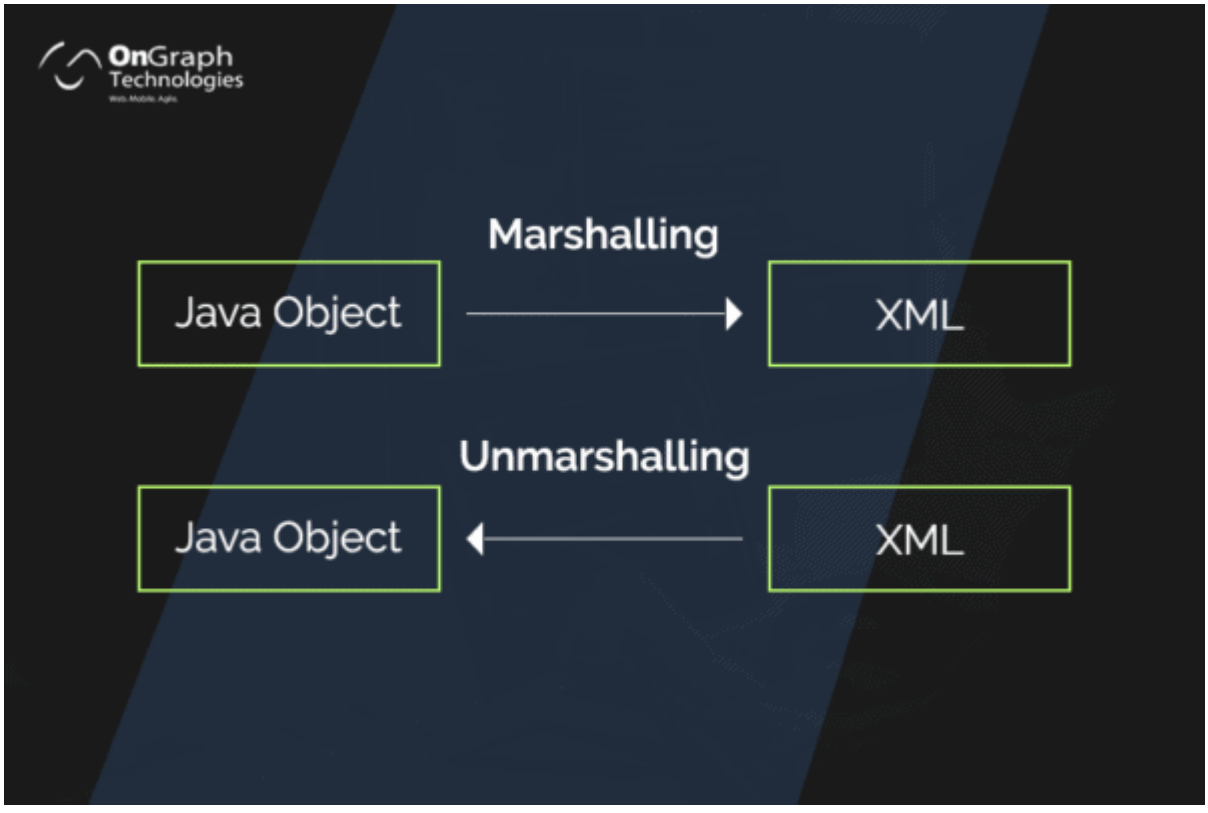
1) What is Heterogeneity?

   In the context of RMI (Remote Method Invocation), heterogeneity refers to the ability of RMI to **support communication between different systems or platforms**. It means that RMI allows distributed objects to interact even if they are running on different hardware architectures, operating systems, or programming languages.

   RMI is designed to enable communication between Java objects across a network. It allows Java objects residing on one system (client) to invoke methods on Java objects residing on another system (server) as if they were local objects. This communication can occur between different Java Virtual Machines (JVMs) running on the same or different physical machines.

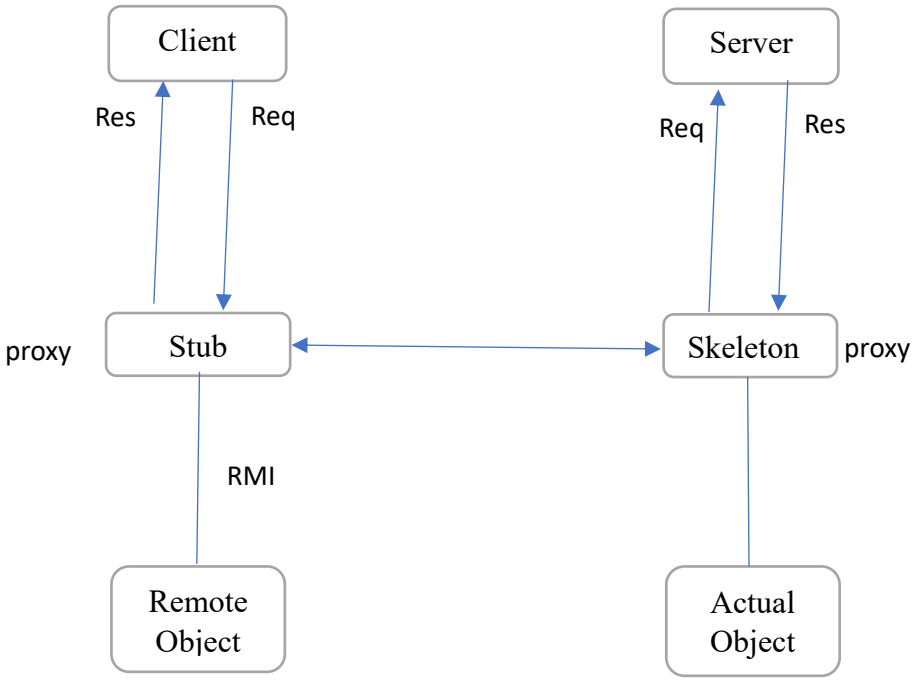
2) Example of is marshalling and unmarshalling?

   Marshalling is the process of converting an object into a format that can be stored or transmitted, while unmarshalling is the process of converting a stored or transmitted format back into an object.



3) Explain RMI with diagram.

- RMI is based on a client-server model.
- RMI is an API that provides a mechanism to create distributed application in JAVA.
- It allows an object to invoke method on an object running in another JVM.
- RMI provides remote Communication between applications using two objects stub and skeleton.
- RMI uses stub and skeleton object in communication with the remote object.



- Remote object is an object whose method can be invoked from another JVM.
- Stub:
  - It's an object, acts as a gateway in the client-side.
  - All outgoing requests are routed through it.
  - It resides at client side and represents the remote object.
- Skeleton:
  - It's an object, acts as gateway in the server side.
  - All the incoming requests are received by this server-side object.

4) What is binding?

Name binding involves associating a meaningful name with a network resource, such as a server, service, or object. This allows clients to refer to the resource using the assigned name instead of knowing its specific network address or location. Name binding simplifies the process of accessing resources in a distributed system and decouples the client from the underlying details of resource location.

RMI Registry: In RMI (Remote Method Invocation), the RMI Registry binds names to remote objects, allowing clients to locate and invoke methods on those objects using a meaningful name.

5) What is role of RMI registry? why we start RMI registry first.

The RMI Registry serves as a central directory for remote objects in Java RMI. It allows objects to register with unique names and provides a lookup mechanism for clients to locate and interact with remote objects. Starting the RMI Registry first enables object registration, lookup, and facilitates communication between clients and remote objects.

6) What is use of "UnicastRemoteObject","lookup(),rebind().

- **UnicastRemoteObject**: "UnicastRemoteObject" is a class in Java RMI that provides functionality for exporting a remote object. By extending this class, a remote object can be made available for remote method invocation. It handles the complexities of communication between the client and the server by providing remote communication support.

- **lookup ()**: The "lookup()" method is used to retrieve a reference to a remote object from the RMI Registry. Clients use this method to locate and obtain a reference to a registered remote object based on its assigned name. It performs a name-based lookup in the RMI Registry and returns the corresponding remote object reference.

- **rebind ()**: The "rebind()" method is used to bind or rebind a remote object to a name in the RMI Registry. It allows a remote object to register itself or update its binding in the registry. If a name is already bound to an object, "rebind()" replaces the existing binding with the new object reference. It is typically used by server-side objects to register themselves with the RMI Registry.

In summary, "UnicastRemoteObject" is used to export a remote object, making it available for remote method invocation. "lookup()" is used by clients to locate and obtain a reference to a registered remote object based on its assigned name. "rebind()" is used by server-side objects to bind or rebind themselves to a name in the RMI Registry, allowing clients to access them through that name.

7) What is stub and skeleton?

- Stub:
  - It's an object, acts as a gateway in the client-side.
  - All outgoing requests are routed through it.
  - It resides at client side and represents the remote object.
- Skeleton:
  - It's an object, acts as gateway in the server side.
  - All the incoming requests are received by this server-side object.

8) What is difference between Exception and RemoteException

Exception:

- Generic term for an abnormal condition or event during program execution.
- Used for error handling, error recovery, and controlling program flow.
- Categorized into checked exceptions (must be declared or caught) and unchecked exceptions (runtime exceptions).
- Examples include "NullPointerException," "IOException," "NumberFormatException," etc.

RemoteException:

- Specific type of exception in Java RMI (Remote Method Invocation).
- Thrown when a remote method invocation or communication between client and server encounters an error.
- Used to handle exceptions related to distributed computing and remote method invocation.
- Checked exception that must be declared or caught when working with RMI.
- Indicates failures or errors in remote method invocation, such as network connectivity issues, server unavailability, serialization/deserialization problems, or communication-related problems.
- Provides information about the cause of the remote method invocation failure for handling and responding to the exception.

Assignment 2

1) What is CORBA?

   CORBA (Common Object Request Broker Architecture) is a standard framework for creating distributed computing systems. It enables different software components (objects) to communicate and interact across networks and programming languages. CORBA uses a middleware called the Object Request Broker (ORB) to handle communication details, and it uses a language-neutral Interface Definition Language (IDL) to define object interfaces. CORBA promotes interoperability and reusability of software components and has been widely used in industries like telecommunications and finance. However, newer technologies like web services and RESTful APIs have gained popularity in recent years.

2) How CORBA works?

   CORBA (Common Object Request Broker Architecture) works through the following steps:

   - **Interface Definition**: Developers define the interfaces of software components using the Interface Definition Language (IDL). The IDL specifies the operations and data types that objects expose to other components.
   - **IDL Compilation**: The IDL is compiled using the CORBA IDL compiler, which generates language-specific stubs and skeletons. Stubs act as proxies for
   - **Object Registration**: Objects are registered with the Object Request Broker (ORB). The ORB maintains a registry of objects, their locations, and the protocols they use for communication.
   - **Client Invocation**: A client application requests a service or invokes a method on a remote object. It does so by calling the appropriate method on the stub object associated with the remote object.
   - **Stub Communication**: The stub intercepts the method invocation and marshals the parameters into a network-neutral format. It then sends the request to the ORB.
   - **Object Location**: The ORB looks up the requested object in its registry and determines its location. It may use various protocols like TCP/IP, IIOP (Internet Inter-ORB Protocol), or proprietary protocols to communicate with the object.
   - **Skeleton Activation**: At the object's location, the ORB activates the skeleton associated with the object. The skeleton unmarshals the parameters and invokes the requested method on the actual object.
   - **Object Processing**: The object performs the requested operation using its internal logic and may access additional services or objects.
   - **Result Marshaling**: The object returns the result to the skeleton, which marshals it into a network-neutral format.
   - **Response Transmission**: The ORB transmits the marshaled result back to the client via the appropriate protocol.
   - **Stub Invocation**: The stub on the client side receives the response, unmarshals it, and returns the result to the client application.

3) What is ORB?

ORB stands for Object Request Broker. It is a crucial component of the CORBA (Common Object Request Broker Architecture) framework. The ORB acts as a middleware, facilitating communication and interaction between distributed software components (objects) in a networked environment.

4) What is IDL interface?

IDL (Interface Definition Language) interface is a key concept in CORBA (Common Object Request Broker Architecture). It is a language-neutral specification that defines the interfaces of software components (objects) within a distributed system.

An IDL interface describes the operations (methods) that an object provides and the data types it uses. It serves as a contract between the client and the server, specifying how they should interact with each other. The IDL interface defines the syntax and semantics of the methods and their parameters, as well as the data structures exchanged between the client and server.

The IDL interface is written in a language-independent manner, allowing it to be understood by different programming languages. It acts as a bridge between the various languages used in the distributed system, enabling seamless interoperability.

5) What is Object Request Broker Daemon (ORBD).

The Object Request Broker Daemon (ORBD) is a specific implementation of the Object Request Broker (ORB) component in the CORBA (Common Object Request Broker Architecture) framework. ORBD is typically provided as part of a CORBA middleware implementation, and it serves as a process or service that runs in the background, managing the communication between distributed objects.

key functionalities:

1. Object Activation
2. Object Location and Registration
3. Communication Management
4. Service Invocation
5. Security and Transaction Support

6) What is middleware.

Middleware refers to software components or services that sit between the operating system and applications, providing a set of functions and services to facilitate

communication and integration between different software systems, applications, or components.

7) List the examples of middleware.
   Examples of Middleware:
   1. Banking
   2. Retail
   3. Healthcare
   4. Manufacturing

8) List the use of middleware.

   Middleware is software that acts as an intermediary between two applications or services to facilitate their communication. It can be used to provide a variety of services, including:

   1. **Data integration**: Middleware can be used to integrate data from different sources, such as databases, files, and web services. This can be useful for building applications that need to access data from multiple sources.
   2. **Application integration**: Middleware can be used to integrate different applications, such as web applications, desktop applications, and mobile applications. This can be useful for building enterprise applications that need to integrate with a variety of systems.
   3. **Security**: Middleware can be used to provide security for applications and data. This can include features such as authentication, authorization, and data encryption.
   4. **Performance**: Middleware can be used to improve the performance of applications. This can include features such as caching, load balancing, and fault tolerance.
   5. **Scalability**: Middleware can be used to make applications more scalable. This can include features such as clustering and distributed processing.

   Middleware can be used in a variety of industries, including:

   1. **Financial services**: Middleware is used in the financial services industry to integrate and secure applications and data. This is important for compliance with regulations such as Sarbanes-Oxley and Dodd-Frank.
   2. **Healthcare**: Middleware is used in the healthcare industry to integrate and secure applications and data. This is important for patient safety and privacy.
   3. **Retail**: Middleware is used in the retail industry to integrate and secure applications and data. This is important for customer service and inventory management.

4. **Manufacturing**: Middleware is used in the manufacturing industry to integrate and secure applications and data. This is important for supply chain management and production planning.
5. **Logistics**: Middleware is used in the logistics industry to integrate and secure applications and data. This is important for tracking shipments and managing inventory.

9) List the applications of CORBA

CORBA (Common Object Request Broker Architecture) has been utilized in various industries and application domains. Some of the applications of CORBA include:

1. **Telecommunications**: CORBA has been extensively used in the telecommunications industry for building distributed systems that integrate different network elements, protocols, and services. It enables interoperability between diverse telecom systems and facilitates the development of scalable and flexible network management solutions.

2. **Finance and Banking**: CORBA has found application in the finance and banking sector for building distributed trading systems, risk management platforms, and transaction processing systems. It allows for seamless communication and integration between different banking applications and services, promoting interoperability and scalability.

3. **Defense and Aerospace**: CORBA has been employed in defense and aerospace applications, where interoperability and integration are crucial. It facilitates communication and interaction between various mission-critical systems, such as command and control systems, radar systems, and communication systems.

4. **Healthcare**: CORBA has been used in healthcare systems to enable interoperability and integration between different medical devices, information systems, and healthcare applications. It allows for seamless communication and exchange of patient data, facilitating efficient healthcare delivery and information sharing.

5. **Manufacturing and Industrial Automation**: CORBA has found application in manufacturing and industrial automation for building distributed control systems, supervisory control and data acquisition (SCADA) systems, and manufacturing execution systems (MES). It enables seamless integration between different components of the manufacturing process, such as sensors, controllers, and management systems.

6. **Enterprise Integration**: CORBA has been utilized in enterprise integration scenarios, where different applications and systems need to communicate and

share data. It allows for seamless integration between enterprise systems, such as customer relationship management (CRM), enterprise resource planning (ERP), and supply chain management (SCM) systems.

7. **Distributed Gaming**: CORBA has been used in the gaming industry for building distributed gaming platforms and multiplayer gaming systems. It enables communication and interaction between game servers, game clients, and other gaming components.

8. **Research and Scientific Computing**: CORBA has found application in research and scientific computing domains, where distributed systems and collaboration are crucial. It facilitates communication and integration between different research instruments, data analysis systems, and scientific computing platforms.

Assignment No. 3

1. What is use of MPI?

MPI (Message Passing Interface) is a standard **library** and specification for **message passing parallel** programming. It provides a programming model for developing **parallel applications** that can run on distributed memory systems.

2. Application in which we are using MPI?

MPI is commonly used in scientific and high-performance computing applications, such as simulations, **weather modeling, computational physics, and data analysis**.

3. Why we are providing rank to process in MPI.

Providing a rank to each process in MPI helps identify and **differentiate individual processes** within a parallel program. The rank serves as a **unique identifier**, allowing processes to communicate and coordinate with each other.

4. Explain MPI operations.

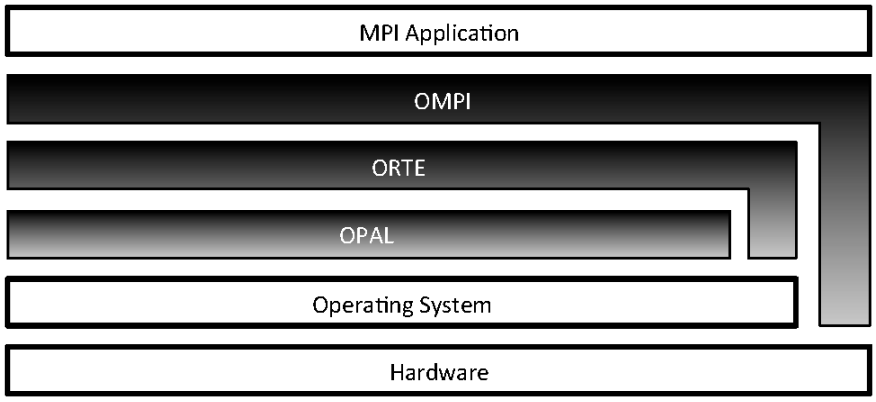
MPI operations refer to the collective **communication** and **computation** functions provided by MPI. These operations enable processes to exchange data, synchronize their execution, and perform computations collectively. Examples include broadcast, reduce, scatter, gather, and barrier operations.

5. Explain Different data types of MPI.

MPI provides various data types to facilitate data exchange and communication between processes. These data types include basic **types (e.g., integers, floating-point numbers), derived types (e.g., structs, arrays), and user-defined type**s (custom data structures).

6. Draw MPI architecture.

The MPI layer is the highest abstraction layer, and is the only one exposed to applications. The MPI API is implemented in this layer, as are all the message passing semantics defined by the MPI standard. Since portability is a primary requirement, the MPI layer supports a wide variety of network types and underlying protocols. Some networks are similar in their underlying characteristics and abstractions; some are not.



7. What is MPI_ABORT?

MPI_ABORT is a function that **terminates** all processes in an MPI **communicator**. It is typically used when an unrecoverable error or exceptional condition occurs, causing the entire parallel program to abort.

8. What is MPI_FINALIZE

MPI_FINALIZE is a function that marks the **end** of an MPI program. It ensures that all pending communication operations are completed before terminating the processes. MPI_FINALIZE should be called after the main computation is finished and before the program exits.

9. What is difference MPI_ABORT and MPI_FINALIZE

The main difference between MPI_ABORT and MPI_FINALIZE is that MPI_ABORT **terminates** all processes in an MPI communicator immediately, while MPI_FINALIZE allows for a **graceful termination** by ensuring completion of pending operations before the processes exit. MPI_ABORT is typically used for error handling, while MPI_FINALIZE is called at the end of a program's execution.

Assignment 4

1) What is the difference between a logical clock and a physical clock?

   Logical Clock:

   - A logical clock is an abstract concept used in distributed systems to establish a partial ordering of events.
   - It is not directly tied to the physical passage of time and does not rely on the accuracy of physical clocks.
   - Logical clocks provide a mechanism to order events that occur in different processes or nodes within a distributed system, even in the absence of a global clock.
   - They are typically implemented using algorithms such as Lamport timestamps or vector clocks.
   - Logical clocks prioritize causality rather than precise time measurements.

   Physical Clock:

   - A physical clock measures the passage of time based on some external reference, such as the oscillations of a quartz crystal or the vibrations of atoms.
   - Physical clocks are typically found in computer systems, wristwatches, or other devices that need to keep track of the real-world time.
   - They provide a representation of absolute time with relatively high accuracy, depending on the quality of the clock and its synchronization with a reference time source (e.g., NTP or GPS).
   - Physical clocks are influenced by factors like clock drift, clock synchronization errors, and latency in signal propagation.
   - They are primarily used for tasks that require precise time measurements, scheduling, or synchronization across different devices.

2) Why is it necessary to synchronize the clocks in distributed real-time system?

   Synchronizing clocks in a distributed real-time system is crucial for several reasons:

   1. **Event Ordering**: In a distributed system, events can occur on different nodes or processes, and establishing a consistent order of events is essential for correct and predictable behavior. Clock synchronization helps determine the causal relationships between events and ensures that events are ordered correctly based on their occurrence.

   2. **Coordination and Scheduling**: Clock synchronization enables effective coordination and scheduling of tasks in a distributed system. Synchronized

clocks provide a common time reference that allows different nodes or processes to coordinate their actions and synchronize their activities based on agreed-upon time intervals or deadlines.

3. **Data Consistency**: In distributed systems, data replication and consistency are often necessary to ensure that multiple replicas of the same data are kept in sync. Clock synchronization helps maintain consistency by providing a reference for determining the ordering of data updates and resolving conflicts when multiple updates occur simultaneously.

4. **Time-Dependent Operations**: Many real-time systems perform operations or make decisions based on time-sensitive criteria. For example, in industrial control systems or financial trading systems, time plays a critical role in determining the correctness and timeliness of actions. Synchronized clocks ensure that time-dependent operations are performed accurately and consistently across the distributed system.

5. **Event Correlation and Debugging**: When diagnosing issues or debugging distributed systems, having synchronized clocks can assist in correlating events and logs across different nodes or processes. With synchronized clocks, it becomes easier to analyze the sequence of events and identify the causes of problems or anomalies.

6. **Communication and Message Ordering**: Clock synchronization is vital for maintaining the consistency and order of messages exchanged between distributed components. Synchronized clocks enable proper message ordering, which is crucial for ensuring reliable communication and preventing issues like message reordering or out-of-order processing.

3) How the principle of the Berkeley algorithm is used to synchronize time in distributed System?

The Berkeley algorithm is a time synchronization algorithm used in distributed systems to achieve clock synchronization among multiple nodes. It was developed by the University of California, Berkeley. The algorithm works as follows:

1. **Selection of Master Node**: One node in the distributed system is designated as the master node or time server. This node is responsible for coordinating the clock synchronization process.

2. **Time Adjustment**: The master node periodically polls the clocks of the other nodes in the system to determine their local times. Upon receiving the responses, the master node calculates the average time of all the nodes.

3. **Time Offset Calculation**: The master node determines the time offset for each individual node by comparing its local time with the average time calculated in the previous step. The time offset represents the difference between the node's clock and the master clock.

4. **Time Adjustment Broadcast**: The master node broadcasts the calculated time offsets to all the other nodes in the system.

5. **Time Adjustment**: Upon receiving the time offsets from the master node, each node adjusts its local clock by applying the respective time offset. This adjustment helps synchronize the clocks of all the nodes with the master clock.

The Berkeley algorithm strives to minimize the time difference among the clocks of the distributed nodes, but it does not aim for high-precision time synchronization. It assumes that the clocks in the system are reasonably accurate but may have some inherent drift or inaccuracy.

4) What are other algorithms for clock synchronization in DS?

In addition to the Berkeley algorithm, several other clock synchronization algorithms have been developed for achieving time synchronization in distributed systems. Some of these algorithms include:

1. **Network Time Protocol (NTP)**: NTP is a widely used clock synchronization protocol that aims to synchronize the clocks of computers over a network. It utilizes a hierarchical model with a few highly accurate time servers acting as primary time sources. NTP employs a combination of clock adjustment and filtering techniques to minimize clock errors and compensate for network delays.

2. **Precision Time Protocol (PTP):** PTP is a more precise clock synchronization protocol designed for systems that require sub-microsecond accuracy. It uses a master-slave architecture, where one node acts as the master clock and others synchronize their clocks with it. PTP employs timestamp-based mechanisms and compensation algorithms to account for clock offsets and network delays.

3. **Global Time Service (GTS)**: GTS is an algorithm that utilizes a centralized time server to distribute time to all the nodes in a distributed system. The time server broadcasts the current time, and each node adjusts its clock accordingly. GTS is suitable for systems where centralized control and synchronization are feasible.

4. **Cristian's Algorithm**: Cristian's Algorithm is an early clock synchronization algorithm that assumes the existence of a time server with an accurate clock. In this algorithm, a client sends a request to the time server, which responds

with the current time. The client estimates the network delay and adjusts its clock accordingly. However, Cristian's Algorithm does not account for variable network delays and assumes a single-time server.

5. **Marzullo's Algorithm**: Marzullo's Algorithm is a fault-tolerant clock synchronization algorithm that aims to achieve consensus on time values in the presence of faulty nodes. It utilizes a voting mechanism where each node proposes its time, and a voting process determines the final agreed-upon time. Marzullo's Algorithm can tolerate failures and inaccuracies in a distributed system.

Assignment 5

1) What is race condition?
   A race condition occurs when the behaviours or outcome of a program depends on the relative timing or interleaving of multiple concurrent threads or processes. It arises when two or more threads access shared resources or perform operations concurrently, and the final result depends on the order of execution, which is unpredictable. Race conditions can lead to unexpected and erroneous behaviour in a program.

2) What is deadlock and starvation?
   Deadlock: Deadlock refers to a situation where two or more threads or processes are blocked forever, waiting for each other to release resources that they hold. In a deadlock, none of the threads can proceed, resulting in a system freeze or unresponsiveness.

   Starvation: Starvation occurs when a thread or process is unable to access the resources it needs to make progress, despite the resources being available. This can happen due to resource allocation algorithms or scheduling policies that prioritize other threads, leading to the starving thread being continuously delayed or bypassed.

3) What is Mutual Exclusion?
   Mutual exclusion is a property or mechanism that ensures that only one thread or process can access a shared resource at any given time. It prevents concurrent access to the resource, thereby avoiding data inconsistency or conflicts that may arise from simultaneous modifications. Mutual exclusion is typically achieved using synchronization constructs, such as locks or semaphores, that allow threads to acquire exclusive access to the shared resource.

4) How to avoid mutual exclusion using
   To avoid mutual exclusion and enable concurrent access to shared resources, you can consider the following techniques:

   - Use fine-grained locking: Instead of locking the entire resource, divide it into smaller independent units and apply locks only on those units that are being modified. This allows multiple threads to access different units simultaneously.
   - Use lock-free or wait-free data structures: These data structures are designed to minimize or eliminate the need for explicit locking, enabling concurrent access without mutual exclusion. They use techniques such as atomic operations, compare-and-swap, or optimistic concurrency control.
   - Use thread-safe data structures: Utilize thread-safe collections or data structures provided by the programming language or libraries. These data structures are

designed to handle concurrent access and ensure data consistency without requiring explicit locking from the developer.

- Employ non-blocking algorithms: Non-blocking algorithms enable multiple threads to make progress concurrently without relying on locks or synchronization. These algorithms use techniques like compare-and-swap or atomic operations to ensure data integrity and progress even in the presence of concurrent modifications.

Assignment No. 6

1. Who is process coordinator? What are its responsibilities?

The process coordinator is a designated process in a distributed system that is responsible for **managing** and **coordinating** the activities of other processes. Its responsibilities include task **allocation, synchronization, communication, and overall system management**.

2. Need of Election Algorithm?

Election algorithms are used in distributed systems to select a **leader or coordinator** among multiple processes. They ensure that a single process is designated as the leader, allowing for efficient coordination and decision-making within the system.

3. What is centralized and decentralized algorithm?

In a centralized algorithm, a central authority or coordinator controls the **election** process. It collects information from all processes and determines the leader. In contrast, a decentralized algorithm allows processes to communicate with each other directly, without relying on a central coordinator for the election process.

4. Explain Election working of algorithm for Ring & Bully?

In the Ring election algorithm, processes are organized in a **logical ring**. Each process sends an **election message** containing its ID to its neighboring processes until the message reaches the highest ID process, which becomes the leader. In the Bully election algorithm, processes with **higher IDs initiate elections by sending election messages to lower ID** processes. If no higher ID process responds, the initiating process becomes the leader.

5. What is “Token”?

In the context of election algorithms, a "token" is a **special message** or token that is passed among processes to facilitate the election process. The process holding the token has the right to initiate an election or become the leader.

6. Why algorithm is known as “Bully”?

The Bully algorithm is known as "bully" because it follows a **hierarchical approach** where higher **ID processes take charge and assert their dominance over lower ID** processes during the election process. The higher ID processes "bully" their way to becoming the leader by sending election messages to lower ID processes.