1. Write short note on following:

(i) Layered Protocol

(ii)Atomic Transaction

(iii)Mutual Exclusion

(iv) Process

(v)DFS

(vi) Shared Memory

Unit-1

2. Define Distributed System. Explain its goals and types

3. What do you understand by communication in distributed system?

Unit-II

4. Describe Election Algorithm in distributed system.

5 Define Deadlock. Also describe deadlock prevention in distributed system.

Unit-III

6. Describe Scheduling and types of scheduling in distributed system.

7. Give a complete description about distributed file system.

Unit-lV

8. Describe Distributed Shared Memory and its types in detail.

9. What is Process Management in MACH?

(i) Layered Protocol

A layered protocol is a design approach in network architecture where communication functions are divided into separate layers. Each layer performs a specific role and interacts with the layers directly above and below it. The OSI (Open Systems Interconnection) model with seven layers (like physical, data link, network, etc.) is a classic example. This approach improves modularity and simplifies troubleshooting and development.

(ii) Atomic Transaction

An atomic transaction is a sequence of operations that are treated as a single unit of work. It either completes entirely or has no effect at all. Atomicity ensures data integrity, particularly in databases and distributed systems, by adhering to the ACID (Atomicity, Consistency, Isolation, Durability) properties.

(iii) Mutual Exclusion

Mutual exclusion is a concept used in concurrent programming to prevent multiple processes from accessing a critical section (shared resource) simultaneously. It ensures that only one process at a time can execute in the critical section, preventing race conditions and ensuring data consistency.

(iv) Process

A process is an instance of a program in execution. It includes the program code, its current activity, and resources such as open files and memory. Operating systems manage processes through process scheduling, creation, and termination, enabling multitasking and efficient CPU use.

(v) DFS (Distributed File System)

A Distributed File System (DFS) allows files to be stored across multiple networked computers while making them appear as if they are located on a single local system. It provides transparency, fault tolerance, scalability, and easy access to shared data across distributed environments.

(vi) Shared Memory
Shared memory is an inter-process communication (IPC) mechanism where multiple processes can access the same memory space. It allows high-speed data exchange and is commonly used in systems requiring fast communication. Synchronization mechanisms (like semaphores) are needed to manage access safely.

Unit-1

2. Define Distributed System. Explain its goals and types

Ans **Definition of Distributed System:**

A **Distributed System** is a collection of independent computers that appear to the users as a single coherent system. These computers communicate and coordinate with each other through a network to achieve a common goal.

Each node (computer) in a distributed system has its own memory and processing power, but they collaborate by exchanging messages.

Goals of Distributed System:

1. **Transparency:**
    - **Access Transparency:** Users should access remote resources like local ones.
    - **Location Transparency:** Users should not need to know the physical location of resources.
    - **Replication Transparency:** Users should be unaware of replicated resources.

   - **Concurrency Transparency:** Multiple users can access resources without interference.
   - **Failure Transparency:** System should recover from failures without affecting users.
2. **Openness:**
   - The system should be extensible and interoperable, using standard protocols and interfaces.
3. **Scalability:**
   - System should work efficiently as the number of users, nodes, or data increases.
4. **Fault Tolerance:**
   - The system should continue operating even if some components fail.
5. **Resource Sharing:**
   - Enables sharing of hardware, software, and data across systems.
6. **Performance:**
   - Distributed systems aim to provide high performance using distributed computing resources.

Types of Distributed Systems:

1. **Client-Server Systems:**
   - Clients request services and servers provide them.
   - Example: Web servers, email servers.
2. **Peer-to-Peer (P2P) Systems:**

    - All nodes are equal and share resources directly.
    - Example: BitTorrent, blockchain networks.
3. **Distributed Computing Systems:**
    - Tasks are divided among multiple nodes for parallel processing.
    - Example: Grid computing, scientific simulations.
4. **Distributed File Systems:**
    - Manage files distributed across multiple machines.
    - Example: Google File System, HDFS.
5. **Distributed Databases:**
    - A database that is spread across multiple sites or nodes.
    - Example: Cassandra, Amazon DynamoDB

3. What do you understand by communication in distributed system?

Ans **Definition:**

Communication in a distributed system refers to the **exchange of data and messages between processes running on different machines**. Since the nodes are physically separated, communication is vital to coordinate tasks and share resources effectively.

Importance of Communication:

- Enables collaboration between distributed components.
- Facilitates resource sharing and process synchronization.
- Supports distributed applications like email, cloud services, and file sharing.

Communication Models:

1. **Message Passing:**
    - Processes exchange information through send and receive operations.
    - Common in socket programming, RPC (Remote Procedure Call).
2. **Remote Procedure Call (RPC):**
    - Allows a program to invoke a function on a remote machine as if it were local.
    - Abstracts the complexities of network communication.
3. **Remote Method Invocation (RMI):**
    - Similar to RPC, but used in object-oriented systems (like Java RMI).
    - Supports invoking methods on remote objects.
4. **Stream-Oriented Communication:**
    - Data is transferred as a continuous stream (e.g., audio/video streaming).
5. **Multicast Communication:**
    - One message is sent to multiple recipients, useful in group communication and broadcasting.

Challenges in Distributed Communication:

1. **Latency and Bandwidth:**
    - Network delays and limited bandwidth can affect communication performance.

2. **Partial Failures:**
    - Communication may fail partially even if some parts of the system are functioning.
3. **Concurrency:**
    - Simultaneous communication needs proper synchronization to avoid conflicts.
4. **Security:**
    - Data transfer over networks is vulnerable; encryption and authentication are essential.
5. **Data Serialization:**
    - Data structures must be converted into a format suitable for transmission (e.g., JSON, XML).

Protocols Used:

- **TCP/IP:** Reliable communication using connection-oriented protocol.
- **UDP:** Faster, but connectionless and unreliable.
- **HTTP/HTTPS:** Widely used for web-based communication.
- **gRPC:** High-performance RPC framework using HTTP/2 and Protocol Buffers.

4. Describe Election Algorithm in distributed system.

Definition:

An **Election Algorithm** in a distributed system is used to designate one process (node) as the **coordinator** or **leader** among several distributed processes. This leader is responsible for managing shared resources, coordinating activities, or making decisions.

When the current leader fails or a new process joins, an election is triggered to select a new coordinator.

Goals of Election Algorithms:

- Elect the process with the **highest priority** (usually the highest ID).
- Ensure **only one leader** is elected.
- **Minimize message passing** and time taken for the election.
- **Detect failures** and recover coordination seamlessly.

Common Election Algorithms:

1. Bully Algorithm:

- **Assumptions:**
  - Each process has a unique ID.
  - Processes can communicate with each other.
  - All processes know the IDs of others.
  - The highest-ID process should become the leader.
- **Steps:**

1. When a process notices the leader is not responding, it initiates an election.
2. It sends **ELECTION** messages to all processes with higher IDs.

3. If no one responds, it becomes the leader and sends **COORDINATOR** messages to all.

4. If a higher process responds, that process takes over the election.

- **Advantages:**
  - Simple and deterministic.
- **Disadvantages:**
  - Can cause many messages in case of many nodes.

---

2. Ring Algorithm:

- **Assumptions:**
  - Processes are arranged in a logical ring.
  - Each process knows its successor in the ring.
  - All have unique IDs.
- **Steps:**

1. A process noticing leader failure sends an **ELECTION** message to its successor.

2. Each process adds its ID to the message and forwards it.

3. When the message returns, the process with the **highest ID** is chosen.

4. A **COORDINATOR** message is sent around the ring announcing the new leader.

- **Advantages:**
  - Works even if processes don’t know all others.

- **Disadvantages:**
    - Slower compared to Bully if the ring is large.

5 Define Deadlock. Also describe deadlock prevention in distributed system.

Ans **Definition of Deadlock:**

A **deadlock** occurs in a distributed system when a group of processes are waiting for each other in a circular chain, and none of them can proceed because they are all waiting for resources held by the others.

Conditions for Deadlock (Coffman Conditions):

1. **Mutual Exclusion** – Only one process can use a resource at a time.
2. **Hold and Wait** – Processes hold resources while waiting for others.
3. **No Preemption** – Resources cannot be forcibly taken.
4. **Circular Wait** – A circular chain of waiting processes exists.

If all these conditions hold simultaneously, a deadlock may occur.

Deadlock in Distributed Systems:

In distributed systems, deadlock is harder to detect and resolve due to:

- Lack of global knowledge.

- No centralized control.
- Communication delays.

Deadlock Prevention Strategies:

Deadlock prevention works by **breaking at least one of the Coffman conditions**.

---

1. Prevent Hold and Wait:

- Require processes to request all resources at once.
- **Drawback:** Leads to low resource utilization and possible starvation.

2. Preempt Resources:

- If a process cannot get all required resources, it releases the held ones.
- The process retries later.
- **Drawback:** High overhead and rollback complexity.

3. Avoid Circular Wait:

- Impose an ordering on resource requests.
- Processes must request resources in a predefined order.
- **Example:** If resources A, B, C are ordered, then a process holding A cannot request C directly.

Other Techniques:

4. Timeout-Based Prevention:

- If a process waits too long, assume deadlock and roll back or restart.

5. Wait-Die & Wound-Wait Schemes (used in distributed databases):

- Use timestamps to decide which process should wait or be aborted to avoid deadlocks.

Unit-III

6. Describe Scheduling and types of scheduling in distributed system.

Ans **Definition of Scheduling:**

Scheduling in a distributed system refers to the decision-making process that determines how and where to execute processes or tasks across multiple computers (nodes). The aim is to optimize resource utilization, reduce response time, and maintain system fairness and efficiency.

Objectives of Scheduling:

- Maximize CPU and resource utilization.
- Minimize response and turnaround time.
- Balance load among nodes.
- Ensure fairness and avoid starvation.
- Adapt to system failures and workload changes.

Types of Scheduling in Distributed Systems:

1. Centralized Scheduling:

- A single scheduler (central authority) decides where tasks are executed.
- It collects information from all nodes before making a decision.

Pros:

- Easier to implement and manage.
- Global view of the system helps in optimal decisions.

Cons:

- Single point of failure.
- Scalability issues as system grows.

---

2. Distributed Scheduling:

- Each node makes its own scheduling decisions.
- Nodes may cooperate to balance load.

Types:

- **Cooperative:** Nodes exchange information to help each other.
- **Non-cooperative:** Each node acts independently.

Pros:

- No single point of failure.
- More scalable.

Cons:

- Decisions may not be globally optimal.
- More communication overhead.

---

3. Static Scheduling:

- Tasks are assigned to nodes **before execution begins**.
- Based on known workloads and resource availability.

Pros:

- Simple and low runtime overhead.
- Useful in predictable environments.

Cons:

- Not adaptive to runtime changes (e.g., node failures).

---

4. Dynamic Scheduling:

- Decisions are made **at runtime**, considering current system state.

- Adapts to load, failures, and changing conditions.

Pros:

- Highly adaptive and flexible.
- Better for heterogeneous and unpredictable systems.

Cons:

- More complex and resource-consuming.

---

5. Load Sharing vs. Load Balancing:

- **Load Sharing:** Transfers tasks only when a node is overloaded.
- **Load Balancing:** Continuously tries to distribute tasks evenly across all nodes.

Scheduling Policies:

- **First-Come-First-Serve (FCFS)**
- **Shortest Job First (SJF)**
- **Priority Scheduling**
- **Round-Robin**
- **Min-Min/Max-Min (used in Grid systems)**

7. Give a complete description about distributed file system.

Ans **Definition:**

A **Distributed File System (DFS)** is a system that allows files to be stored and accessed across multiple networked computers, while appearing to users and applications as if they are stored locally on a single machine.

Goals of DFS:

- Provide **transparent access** to remote files.

- Ensure **location, access, and replication transparency**.
- Enable **resource sharing** across geographically distributed systems.
- Offer **reliability, fault tolerance**, and **scalability**.

Key Features of DFS:

1. **Transparency:**
    - **Access Transparency:** User sees a unified file namespace.
    - **Location Transparency:** Users don't need to know where a file is stored.
    - **Replication Transparency:** Multiple copies can exist for reliability.
    - **Concurrency Transparency:** Multiple users can access files without conflicts.
2. **Fault Tolerance:**
    - File replication, error detection, and automatic failover support.
3. **Consistency:**
    - Ensures all users see the most recent version of a file.
4. **Security:**
    - Authentication, authorization, and encryption mechanisms.
5. **Scalability:**

    - Supports large number of users and vast amounts of data across nodes.

Architecture of DFS:

1. **Client-Server Model:**
    - Clients request file access; servers manage file storage.
    - Example: NFS (Network File System)
2. **Peer-to-Peer DFS:**
    - All nodes act as both clients and servers.
    - Example: IPFS (InterPlanetary File System)
3. **Multi-server DFS:**
    - Several servers handle file storage; metadata may be centralized or distributed.

Components of DFS:

- **Metadata Server:** Manages file names, permissions, and directory structure.
- **Data Server:** Stores the actual file content.
- **Client Module:** Interface that allows local applications to access distributed files.

Examples of DFS:

1. **NFS (Network File System)** – Sun Microsystems' client-server based DFS.

2. **AFS (Andrew File System)** – Secure and scalable DFS developed at CMU.
3. **HDFS (Hadoop Distributed File System)** – High-throughput DFS for big data.
4. **Google File System (GFS)** – DFS designed for Google's large-scale data processing.

Challenges in DFS:

- **Consistency management** across replicas.
- **Fault tolerance** during server or network failure.
- **Security** against unauthorized access and data breaches.
- **Latency** in file access due to remote communication.

Benefits of DFS:

- Simplifies data access for users and applications.
- Increases data availability and redundancy.
- Enables collaboration and data sharing in distributed environments.

Unit-lV

8. Describe Distributed Shared Memory and its types in detail.

Ans **Definition:**

Distributed Shared Memory (DSM) is an abstraction that allows processes running on different computers in a distributed system to **share a common logical memory space**, even though the physical memory is distributed across multiple nodes.

It allows programmers to treat distributed memory as if it were centralized, simplifying development by **hiding communication details**.

Objectives of DSM:

- Provide a **shared memory abstraction** on top of a distributed system.
- Hide complexities of message passing.
- Allow processes to **share data seamlessly**.
- Support **scalability**, **portability**, and **transparency**.

Characteristics of DSM:

1. **Location Transparency** – Processes don't know where data is stored.
2. **Access Transparency** – Reading and writing is same as local memory.
3. **Consistency Model** – Ensures all nodes see a consistent view of memory.
4. **Replication** – Improves fault tolerance and performance.

Architecture of DSM:

- **Manager-based DSM:** Each memory block has a manager node responsible for tracking and updating it.
- **Home-based DSM:** Each memory page has a designated home node.
- **Peer-to-Peer DSM:** All nodes share responsibilities equally.

Types of Distributed Shared Memory:

1. Based on Memory Granularity:

- **Page-based DSM:**
  - Memory is divided into fixed-size pages.
  - Similar to virtual memory in OS.
  - Efficient but may lead to **false sharing** (when unrelated data on the same page is accessed by different processes).
- **Object-based DSM:**
  - Shared units are data objects.
  - Easier to manage, avoids false sharing.

    - More suitable for object-oriented applications.
- **Variable-size Block DSM:**
    - Memory is divided into logical segments or blocks of varying size.
    - Provides a balance between flexibility and complexity.

---

2. Based on Consistency Models:

- **Strict Consistency:**
    - Every read returns the latest written value.
    - Very difficult to implement in distributed systems.
- **Sequential Consistency:**
    - Operations appear to execute in a consistent global order.
    - Order of operations by a single process is preserved.
- **Causal Consistency:**
    - Writes that are causally related must be seen in the same order.
    - Independent writes can be seen in different orders.
- **Release Consistency:**
    - Synchronization (acquire/release) is used to enforce consistency.
    - More efficient than sequential consistency.
- **Eventual Consistency:**
    - All copies of data will become consistent over time.
    - Common in systems like NoSQL databases (e.g., Amazon DynamoDB).

Advantages of DSM:

- Simplifies programming model by abstracting communication.
- Encourages **data sharing** across processes.
- Improves **portability** and **modularity**.

Disadvantages:

- Performance overhead due to **consistency maintenance**.
- Difficult to **debug** due to hidden communication.
- Not suitable for highly dynamic environments (e.g., real-time systems).

9. What is Process Management in MACH?

Ans **Introduction to MACH:**

MACH is a **microkernel-based operating system** developed at Carnegie Mellon University. It was designed to support **multiprocessing**, **distributed systems**, and **message-based communication**.

One of the key features of MACH is its advanced **process management**, which differs from traditional UNIX systems.

Definition of Process Management in MACH:

In MACH, **process management** is responsible for **creating, managing, and terminating tasks (processes) and threads**, while supporting **inter-process communication (IPC)** and resource allocation.

Key Concepts in MACH Process Management:

1. Tasks and Threads:

- **Task:**
  - A task is a container for resources such as virtual memory and ports.
  - Analogous to a UNIX process but more flexible.

- **Thread:**
    - The unit of execution within a task.
    - A task can have multiple threads, each with its own execution context (PC, stack, registers).

✅ A task = memory + resources; a thread = execution.

---

2. Port-based Communication (IPC):

- MACH uses **message passing** via **ports** for communication between tasks.
- Ports are secure communication channels.
- Tasks communicate by sending/receiving messages to/from ports.

---

3. Process Creation:

- **task_create** and **thread_create** system calls are used to create new tasks and threads.
- Unlike traditional UNIX fork(), MACH separates address space (task) from execution (thread).

---

4. Virtual Memory Management:

- Each task has its own **virtual address space**.
- Supports **memory mapping**, **copy-on-write**, and **demand paging**.

---

5. Scheduling:

- MACH provides **preemptive**, **priority-based scheduling**.
- Each thread is scheduled independently.
- Supports multiprocessor systems by running multiple threads in parallel.

---

6. Thread Management:

- Threads are lightweight and share the task's memory.
- Each thread can be individually scheduled.
- System calls for thread control:
    - thread_resume, thread_suspend, thread_terminate.

Advantages of MACH's Process Management:

- **Fine-grained control** through tasks and threads.
- **Efficient inter-process communication** using ports.
- **Better performance on multiprocessor systems**.
- Flexible and **modular kernel** design.

Limitations:

- Performance overhead due to message passing.
- Higher complexity compared to monolithic kernels.
- Requires more system resources for IPC and thread handling.