Yashoda Shikshan Prasarak Mandal's

# Yashoda Technical Campus, Satara

(Approved by AICTE, Delhi /Approved by Govt of Maharashtra DTE)
Email: principalengg_ytc@yes.edu.in, Web : www.yes.edu.in
NH-4, wadhe,Satara,Tele Fax-02162-271238/39/9172220775

**Faculty of Engineering**
**Department of Computer Science and Engineering**

---

**A.Y: 2024-25 (Sem- II)**
**Class: FY – Mtech. CSE**
**Course code and Name: MTCSEPE203C Program Elective-II**
**Distributed System Principle**
**Course Coordinator: Dr. S V Balshetwar**

**Unit-III:** 3 Marks Questions

## 🔹 Q1 (Level 3 – Apply):

**Apply the Ricart-Agrawala algorithm to explain how three processes coordinate access to the critical section.**

**Solution:**
Each process sends a REQUEST message with its timestamp to the other two.

- A process enters its **critical section (CS)** only after receiving REPLY messages from both.
- After exiting the CS, it sends REPLY to any pending requests.
  This ensures mutual exclusion using **timestamp-based ordering** and **direct communication**.

---

## 🔹 Q2 (Level 3 – Apply):

**Apply token-based mutual exclusion in a ring topology with 4 processes. Describe how token passing ensures exclusive access.**

**Solution:**

- The token circulates among P1 → P2 → P3 → P4 → P1.
- A process enters the CS **only when it holds the token**.
- If a process doesn't need the CS, it simply forwards the token.
  This ensures only one process accesses the CS at any time (mutual exclusion).

---

## 🔹 Q3 (Level 3 – Apply):

**Apply the concept of performance metrics to evaluate a mutual exclusion algorithm with high message overhead.**

**Solution:**

- Analyze message complexity (e.g., Ricart-Agrawala uses 2*(n−1) messages).

- High message count increases **network traffic** and **latency**.
- Suggest a **token-based algorithm** with fewer messages per CS entry (1 message/token).

---

### 🔹 Q4 (Level 4 – Analyze):

**Analyze the drawbacks of centralized mutual exclusion when the coordinator crashes.**

**Solution:**

- The coordinator is a **single point of failure**.
- When it crashes, all processes are **blocked from entering CS**.
- Recovery is non-trivial—requires **new coordinator election** and **state restoration**. This limits reliability and fault tolerance.

---

### 🔹 Q5 (Level 4 – Analyze):

**Analyze and compare fairness in Ricart-Agrawala vs Token Ring algorithms.**

**Solution:**

- **Ricart-Agrawala**: Ensures fairness through **timestamp comparison**, processes with earlier requests are prioritized.
- **Token Ring**: Provides fairness via **round-robin scheduling**, but can delay access if token travels long.
- Both prevent starvation but differ in latency and message usage.

---

### 🔹 Q6 (Level 4 – Analyze):

**A distributed system experiences delays in entering CS. Analyze which performance metric is being affected and why.**

**Solution:**

- The affected metric is **Synchronization Delay**.
- Causes may include **network latency**, **message processing time**, or **high contention** for CS.

- Performance is impacted as processes **wait longer** despite fairness being maintained.

---

## Unit II- 3 marks Question

### 🔹 Q1 (Level 3 – Apply):

**Apply Lamport's logical clock rules to the following events and assign logical timestamps:**

- P1: e1, e2
- P2: f1, f2
- Message m is sent from e2 to f2.

**Solution:**

- P1: e1 = 1, e2 = 2
- Message m sent from e2 = 2
- On receiving m, P2 updates clock: f2 = max(clock at f1 + 1, received timestamp + 1)
- If f1 = 1, then f2 = max(2, 3) = 3
  **Timestamps:**
- e1 = 1, e2 = 2
- f1 = 1, f2 = 3

---

### 🔹 Q2 (Level 3 – Apply):

**Apply vector clocks to determine causality. Given:**

- VC(A) = [2, 0], VC(B) = [1, 1]
  Determine if the events are causally related or concurrent.

**Solution:**
Compare element-wise:

- [2, 0] vs [1, 1] → A > B in first element, A < B in second
  → No total ordering
  **Conclusion:** Events are **concurrent**

---

---

### 🔹 Q3 (Level 3 – Apply):

**Apply the concept of physical clocks in synchronizing distributed systems using Cristian's algorithm. Describe the basic idea.**

**Solution:**

- A client requests time from a time server
- Server replies with its time
- Client adjusts local clock = server_time + ½(RTT)
- Assumes symmetric delay
  Used to **synchronize physical clocks** in distributed systems.

---

### 🔹 Q4 (Level 4 – Analyze):

**Analyze how Lamport's logical clocks fail to capture causality completely.**

**Solution:**

- Lamport timestamps ensure **happened-before relation** (if a → b, then L(a) < L(b))
- But **L(a) < L(b)** doesn't imply a → b
- Can't distinguish **concurrent events**
  Thus, Lamport clocks **preserve order but not causality**.

---

### 🔹 Q5 (Level 4 – Analyze):

**Analyze how vector clocks solve the limitations of Lamport clocks.**

**Solution:**

- Vector clocks use a **vector of counters**, one per process
- Allow detection of **causal relationships** by comparing vectors element-wise
- Can distinguish between **a → b** and **a || b** (concurrent)
  Hence, vector clocks are more expressive for causality tracking.

---

---

### 🔹 Q6 (Level 4 – Analyze):

**Analyze how the Chandy-Lamport algorithm ensures a consistent global state in a distributed system.**

**Solution:**

- **Marker messages** are used to initiate snapshot
- Each process records its state and the **state of incoming channels**
- Messages in transit are captured
  This guarantees a **consistent global snapshot** without stopping the system.

---

## Unit -I - 3 marks Question

### 🔹 Q1 (Level 3 – Apply):

**Apply your understanding of RISC and CISC architectures to justify which one is better suited for real-time embedded systems.**

**Solution:**

- RISC uses a **smaller set of simple instructions** that execute faster.
- It has **faster instruction execution and lower power consumption**.
- For real-time systems requiring speed and predictability, **RISC is more suitable**.

---

### 🔹 Q2 (Level 3 – Apply):

**Apply the OSI model to explain which layers are involved when a user accesses a website using a browser.**

**Solution:**

- **Application Layer**: Handles HTTP requests.
- **Transport Layer**: Ensures reliable delivery (TCP).
- **Network Layer**: Handles IP addressing and routing.
  These layers work together to **deliver and manage web content to the user.**

---

### 🔹 Q3 (Level 3 – Apply):

**Apply the concept of multi-core processing to explain how distributed applications can achieve better performance.**

**Solution:**

- Multi-core processors enable **parallel execution of threads**.
- Distributed applications can be divided into tasks that run concurrently.
- This leads to **faster execution and resource optimization**.

---

### 🔹 Q4 (Level 4 – Analyze):

**Analyze how the evolution of operating systems has contributed to the development of distributed systems.**

**Solution:**

- Early OSes were single-user and single-tasking.
- Modern OSes support **multiprocessing, networking, and virtualization**.
- This evolution allows **process migration, remote execution, and resource sharing**—key for distributed systems.

---

### 🔹 Q5 (Level 4 – Analyze):

**Analyze how different types of transparency improve user experience in a distributed computing system.**

**Solution:**

- **Location transparency**: Users access resources without knowing their location.
- **Access transparency**: Same operations for local and remote resources.
- **Failure transparency**: System recovers from failures without user intervention.
  These transparencies provide a **seamless and consistent user experience**.

---

### 🔹 Q6 (Level 4 – Analyze):

**Analyze the fundamental design goals of distributed systems and explain how they affect system performance.**

**Solution:**

- **Scalability**: Ensures the system handles growth.
- **Fault tolerance**: Maintains operation despite failures.
- **Resource sharing**: Improves efficiency.
  Meeting these goals enhances **availability, reliability, and responsiveness** of distributed systems.

---

## Unit III -8 marks question

### 🔹 Q1 (Level 3 – Apply):

**Q:** Apply the Ricart-Agrawala algorithm for mutual exclusion to a system of 3 processes (P1, P2, P3). Show how P2 gains access to the critical section and how messages are exchanged.

**Solution:**

1. **P2 wants to enter the critical section**:
   - o It timestamps the request and sends it to P1 and P3.
2. **P1 and P3 receive the request**:
   - o If not interested or P2's timestamp is earlier, they send **REPLY** to P2.
   - o If they are also requesting the CS, they compare timestamps and may defer reply.
3. **P2 enters the CS** only after receiving **REPLY from both P1 and P3**.
4. After completing its CS, P2 sends any **deferred REPLY messages** to others.

**Messages exchanged:**

- 2 REQUESTs from P2
- 2 REPLYs to P2
  ➡️ **Total: 4 messages** (for P2 to enter CS)

---

## 🔹 Q2 (Level 4 – Analyze):

**Q:** Analyze the trade-offs between token-based and permission-based distributed mutual exclusion algorithms. Provide a comparative table and discuss performance metrics.

**Solution:**

| Feature | Token-Based | Permission-Based (e.g., Ricart-Agrawala) |
|---|---|---|
| **Messages per CS entry** | 1 (token passing) | 2(n−1) request/reply messages |
| **Fairness** | Fair (token circulates) | Fair (based on timestamps) |
| **Fault Tolerance** | Token loss = recovery needed | Coordinator/process crash affects replies |
| **Synchronization Delay** | Low | Medium to High |
| **Starvation** | Prevented if token circulates | Prevented if processes respond fairly |

- **Analysis**:
  - **Token-based** is efficient under low contention.
  - **Permission-based** is better if message loss is rare and requires no token recovery.
  - Choice depends on network reliability and number of processes.

---

## 🔹 Q3 (Level 4 – Analyze):

**Q:** Analyze the performance of centralized, token-based, and Ricart-Agrawala mutual exclusion algorithms using metrics like message complexity, delay, fault tolerance, and scalability.

**Solution:**

| Algorithm | Msgs per Entry | Delay | Fault Tolerance | Scalability |
|---|---|---|---|---|
| **Centralized** | 3 | Low | Poor (single point fail) | Moderate |
| **Token-Based** | 1 | Low | Needs token recovery | High |

| Algorithm | Msgs per Entry | Delay | Fault Tolerance | Scalability |
|---|---|---|---|---|
| **Ricart-Agrawala** | 2(n−1) | Medium | Better than centralized | Limited by n |

- **Conclusion**:
  - **Centralized**: Simple, low overhead, but not fault-tolerant.
  - **Token-Based**: Scales well but needs token management.
  - **Ricart-Agrawala**: Reliable in permission-based systems, but more message-heavy.

---

### 🔹 Q4 (Level 3 – Apply):

**Q:** Apply the concept of performance metrics to evaluate a distributed mutual exclusion algorithm in a system of 5 processes. Consider metrics: message complexity, response time, and synchronization delay.

**Solution:**

Assume the use of **Ricart-Agrawala algorithm**:

- **Message Complexity** = 2(n−1) = 2 × 4 = 8 messages per CS entry
- **Response Time** depends on message delay + queue wait → Higher under load
- **Synchronization Delay** = Time to receive all REPLY messages
  → Delay increases with **network latency and contention**

**Evaluation:**

- Efficient in small systems
- Message overhead grows with n
- Not ideal for high-frequency CS access or large networks

**Conclusion:** Use token-based or quorum-based algorithms when system scales or latency becomes critical.

---

## Unit -II 8 marks question

### 🔹 Q1 (Level 3 – Apply):

---

**Q:** Apply Lamport's logical clock algorithm to a system of 3 processes (P1, P2, P3) with the following events:

- P1: a → b → sends m1 to P2
- P2: c → receives m1 → d → sends m2 to P3
- P3: receives m2 → e

Assign Lamport timestamps to each event.

**Solution:**
**Lamport's Rules:**

1. Increment local clock before each event.
2. On receiving a message, set clock = max(local, received) + 1.

**Step-by-step:**

- P1:
  - a = 1
  - b = 2
  - send m1 = 3
- P2:
  - c = 1
  - receive m1 → max(1, 3) + 1 = 4
  - d = 5
  - send m2 = 6
- P3:
  - receive m2 → max(0, 6) + 1 = 7
  - e = 8

**Final Timestamps:**

- P1: a=1, b=2, send=3
- P2: c=1, receive=4, d=5, send=6
- P3: receive=7, e=8

---

## 🔹 Q2 (Level 4 – Analyze):

**Q:** Analyze the differences between Lamport clocks and vector clocks in capturing causal relationships. Give an example to justify your answer.

**Solution:**

| Feature | Lamport Clocks | Vector Clocks |
|---|---|---|
| Type | Scalar | Vector (size = number of processes) |
| Ordering Capability | Partial (a → b ⇒ L(a) < L(b)) | Total (can detect concurrency) |
| Causal Relationship | Cannot detect concurrency | Can detect both causal and concurrent events |

**Example:**

Let P1 send to P2 and P3 perform independent events.

- Lamport:
  - P1: a = 1, send m1 = 2
  - P2: receive m1 = 3
  - P3: e = 2
  - L(e) = 2, L(receive) = 3 → L(e) < L(receive) → falsely assumes e → receive
- Vector:
  - P1 VC: [1, 0]
  - P2 after receiving: [1, 1]
  - P3 VC: [0, 0, 1]
  - VC(P3) not ≤ VC(P2) → events are **concurrent**

Vector clocks provide **more accurate causal tracking** than Lamport clocks.

---

## Q3 (Level 3 – Apply):

**Q:** Apply Chandy-Lamport's snapshot algorithm to record a consistent global state in a system with 3 processes communicating over channels. Explain the steps and state how channel messages are handled.

**Solution:**

**Steps:**

1. **Initiator** records local state and sends **marker** to all outgoing channels.
2. On receiving a **marker** for the first time:
   - o Record local state
   - o Send marker to other outgoing channels
   - o Start recording messages on other incoming channels
3. On receiving **subsequent markers**:
   - o Stop recording that channel
   - o Messages recorded between receiving the first marker and subsequent marker are considered **in-transit**

**Application:**

- P1 initiates snapshot
- Records its state
- Sends markers to P2, P3
- P2 and P3 record local states and start recording incoming messages until they receive marker on all channels.

Snapshot represents a **consistent global state** including in-transit messages.

---

### 🔹 Q4 (Level 4 – Analyze):

**Q:** Analyze the use of physical clocks in distributed systems. How do synchronization algorithms (e.g., Cristian's or Berkeley's) address clock drift and skew?

**Solution:**

**Issues with Physical Clocks:**

- **Clock skew**: Difference in clock times among systems
- **Clock drift**: Clocks run at slightly different speeds
- Causes inconsistencies in timestamps, logs, and event ordering

**Cristian's Algorithm:**

- Client requests time from time server
- Adjusts for estimated network delay
- Limited by **network latency asymmetry**

**Berkeley's Algorithm:**

- Master polls all clocks
- Computes average
- Sends adjustments to all nodes
- Works well in systems without external reference (e.g., no GPS)

**Conclusion:**

- Synchronization minimizes skew
- Physical clocks alone are unreliable
- These algorithms **approximate a global clock** essential for coordination

---

## Unit I - 8 marks question

### 🔹 Q1 (Level 3 – Apply):

**Q:** Apply your understanding of CISC and RISC architectures to compare their performance in distributed computing environments. Provide real-world examples where applicable.

**Solution:**

- **CISC (Complex Instruction Set Computing):**
  - Uses **complex instructions** that execute multiple tasks in a single instruction.
  - Example: x86 architecture used in traditional desktop systems.
  - Suitable for systems with limited memory but high instruction flexibility.
- **RISC (Reduced Instruction Set Computing):**
  - Uses **simple, uniform instructions** that execute quickly.
  - Example: ARM architecture used in mobile and embedded devices.
  - Favoured in distributed systems requiring **speed, power efficiency**, and **parallelism**.
- **Application in Distributed Systems:**
  - **RISC** processors in edge devices and IoT nodes.
  - **CISC** may be used in backend servers with complex computational needs.
  - RISC is preferred where **low latency and high throughput** are critical.

---

### 🔹 Q2 (Level 3 – Apply):

**Q:** Apply the ISO/OSI model to explain how communication occurs in a distributed file-sharing application. Identify the relevant layers and their roles.

**Solution:**
In a distributed file-sharing system, layers involved:

1. **Application Layer**:
   - Interfaces with the user and application (e.g., HTTP, FTP).
   - Manages file requests and responses.
2. **Presentation Layer**:
   - Handles **data encoding**, compression, and encryption.
3. **Session Layer**:
   - Manages **sessions and connections** between nodes.
4. **Transport Layer**:
   - Ensures **reliable data delivery** (e.g., TCP), error recovery.
5. **Network Layer**:
   - Provides **routing and logical addressing** (e.g., IP).
6. **Data Link Layer**:
   - Handles **frame creation**, MAC addressing, error detection at link level.
7. **Physical Layer**:
   - Transmits raw bits via cables, wireless signals.

**Conclusion:**
The ISO/OSI model ensures **modular, reliable, and scalable communication** in distributed systems.

---

### 🔹 Q3 (Level 4 – Analyze):

**Q:** Analyze how the evolution of operating systems has supported the development of distributed systems. Include key milestones and their impact.

**Solution:**

| Evolution Stage | Contribution to Distributed Systems |
|---|---|
| **Batch Systems** | Limited – no interactivity or resource sharing |
| **Multiprogramming** | Allowed multiple processes – essential for concurrency |
| **Time-Sharing Systems** | Enabled **remote login**, basic **networking features** |
| **Network OS** | Added support for **communication between systems** |
| **Distributed OS** | Provided **single-system image**, **resource transparency** |

**Impact:**

- OS now supports **remote process execution**, **file sharing**, **inter-process communication (IPC)**.
- Modern OS (e.g., Linux, Windows Server) includes **thread libraries, socket APIs, and virtualization** tools critical for distributed computing.

**Conclusion:**
OS evolution from standalone execution to **networked and distributed operation** has been central to enabling today's distributed systems.

---

### 🔹 Q4 (Level 4 – Analyze):

**Q:** Analyze the design goals and transparency types in distributed computing systems. How do they contribute to system performance and user experience?

**Solution:**

**Design Goals of Distributed Systems:**

- **Transparency**: Hide system complexity.
- **Openness**: Standard interfaces, interoperability.
- **Scalability**: Ability to grow (e.g., horizontally).
- **Fault Tolerance**: Continue service despite failures.
- **Resource Sharing**: Maximize hardware/software utilization.

**Types of Transparency:**

| Type | Description |
|---|---|
| **Location** | Hide where resources are located |
| **Access** | Uniform access to local/remote resources |
| **Replication** | Users unaware of resource duplication |
| **Concurrency** | Multiple users can access same resources |
| **Failure** | System masks failure recovery |
| **Migration** | Resources/processes move without affecting performance |

**Contribution to Performance/User Experience:**

- **Improved usability** through seamless access.

- **Increased reliability** by masking failures.
- **Optimized performance** through load balancing and replication.

---