AUTUMN MID SEMESTER EXAMINATION-2024

School of Computer Engineering
Kalinga Institute of Industrial Technology, Deemed to be University
Distributed Operating System
[CS30009]

Time: 1 1/2 Hours **Full Mark: 20**

Sample answer & evaluation scheme

Answer Any four questions including question No.1 which is compulsory.
The figures in the margin indicate full marks.
Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

1. Answer all the questions. [1 Mark X 5]

a) Designing a multicomputer system is relatively easier than a multiprocessor system. (True/False). Justify your statement.

Answer: False. **[.5]**

Justification: **[.5]**

i. Multiprocessors share a common memory space. Hence communication and data sharing is easy in this environment
ii. Multiprocessor synchronization is easier than multicomputer synchronization due to distributed memory.
iii. With respect to scalability, multicomputer systems is more complex due to network delay, data distribution, and suitable network protocol.
iv. Design of network, fault tolerance, data flow redundancy makes multicomputer systems and its required protocol complex.

b) Why is avoiding centralized components crucial for scalability in distributed systems?

Answer: Centralized components can become bottlenecks, limiting the system's ability to scale and handle increased loads. Decentralizing components allows the system to grow more efficiently.

c) Differentiate between tightly coupled systems and loosely coupled systems.

Scheme of evaluation: At least two features should be mentioned [.5 + .5]

Answer: The difference between tightly coupled systems and loosely coupled systems with respect to different parameters are as below:

Features	**Tightly Coupled Systems**	**Loosely Coupled Systems**
Architecture	Multiple processors integrated into a single physical unit with shared memory.	Multiple independent computers interconnected via a network with their own local memory
Communication	By shared memory	Using message passing
Synchronization	Using locks, semaphores and monitors	Requires explicit messaging protocols
Performance	Lower latency	Communication cost is more
Scalability	Challenging, performance degrades	Adding suitable network devices it would be scalable well.

d) What issue occurs in a non-blocking receive() primitive. How to fix it?

Scheme of evaluation: Both the features should be mentioned [.5 + .5]

Answer: The issue in a non-blocking receive() primitive is when a message arrives in the message buffer, how does the receiving process know?

One of the following two procedures can be used for this purpose:

Polling: In the polling method, the receiver can check the status of the buffer when a test primitive is passed in this method.

The receiver regularly polls the kernel to check whether the message is already in the buffer.

Interrupt: A software interrupt is used in the software interrupt method to inform the receiving process regarding the status of the message i.e. when the message has been stored into the buffer and is ready for usage by the receiver.

e) What is the consequence of a process holding the token, but does not need to enter into the critical section?

Answer: When a process holds the token but does not require the critical section, then the token is simply passed to the next process in the ring, using a token passing mechanism. This ensures that the token circulate1s and eventually reaches processes that require access to the critical section.

2. **(a) How does fault tolerance affect the performance and reliability of distributed systems?**
[2.5 Marks]

Scheme of evaluation: Defining fault tolerance: 0.5 marks
Impact on performance: 1.0 marks
Impact on Reliability: 1.0 marks

Answer: The fault tolerance of a system affects its performance and its reliability to some extent. It is described below:

Fault tolerance is a crucial design principle in distributed systems that significantly enhances both performance and reliability. By definition, fault tolerance refers to the system's ability to continue operating correctly even when some of its components fail. This capability is vital in distributed systems due to their inherent complexity and the higher likelihood of partial failures, such as node crashes, network issues, or data corruption.

Impact on Reliability

Continuous Operation: Fault tolerance ensures that the system remains available and operational even when some parts fail. This continuous operation is critical for maintaining user trust and meeting service level agreements (SLAs), especially in systems that require high availability, such as financial services, e-commerce platforms, and cloud computing environments.

Data Integrity: By using redundancy and replication, fault-tolerant systems safeguard against data loss. For example, in a distributed database, data might be replicated across multiple nodes. If one node fails, the system can retrieve the data from another node, ensuring that no data is lost and maintaining consistency across the system.

Reduced Downtime: A fault-tolerant system minimizes downtime by quickly recovering from failures. Techniques like failover (automatically switching to a standby system) and rollback (reverting to a previous stable state) allow the system to restore service promptly, thereby minimizing the impact of failures on users and business operations.

Impact on Performance

Resource Overheads: Implementing fault tolerance often involves maintaining redundant components, such as duplicate servers or data copies, which can increase resource usage. While this redundancy ensures reliability, it also means that more computational power, memory, and storage are consumed, potentially impacting the system's overall performance.

Latency and Throughput: Fault tolerance mechanisms, such as replication and consensus protocols (e.g., Paxos, Raft), can introduce additional latency because of the need to synchronize data across multiple nodes. This synchronization ensures that all replicas have the latest data, but it can slow down transaction processing and reduce throughput.

Complexity in System Design: Achieving fault tolerance requires sophisticated algorithms and system architectures, which can complicate system design and maintenance. This complexity might result in longer development times and more intricate debugging processes, potentially affecting the system's efficiency and performance.

(b) Compare, and contrast between switch-based multiprocessors and switch-based multicomputers with separate examples for each type. [2.5 Marks]

Scheme of evaluation: [1.0 +1.0 + 0.5] marks
Switched-based multiprocessor: 1.0 mark
Switched-based multicomputers: 1.0 mark
Example: 0.5 mark

Answer:

Switch-Based Multiprocessor:

Architecture: In a switch-based multiprocessor, multiple processors are connected to a shared memory via a switch that handles communication and memory access requests. All processors share the same memory space, allowing for direct access to shared data.

Example: A high-performance computing cluster where multiple CPUs share RAM through a network switch. The communication may involve using protocols like MPI (Message Passing Interface) for coordination among processes, but fundamentally, they operate in a shared memory environment.

Advantages:

- Easier data sharing and consistency due to shared memory.
- Faster communication among processors as they access the same memory.

Switch-Based Multicomputer:

Architecture: In a switch-based multicomputer, each computer (or node) operates independently with its local memory and communicates through a network switch. The nodes do not share memory, and communication occurs via message passing.

Example: A distributed computing cluster where each node (a separate machine) communicates over a switch (such as Ethernet switches) using message-passing protocols. Each node runs its operating system and has its memory.

Advantages:

- Greater scalability, as new nodes can be added without the need for shared memory coherence.
- Better fault isolation; if one node fails, others can continue to operate independently.

Comparison Summary:

Features	**Switched-based Multiprocessor**	**Switched-based Multicomputer**
Memory Access	Shared memory	Local memory
Communication Method	Direct memory access	Message passing
Scalability	Limited by shared memory architecture	High scalability with independent nodes
Fault Tolerance	Limited; failure can affect shared memory	Better; Individual nodes may fail without affecting others
Example	High performance server with shared RAM	Cluster of independent servers for parallel processing in distributed system environment

In summary, the choice between a switch-based multiprocessor and a switch-based multicomputer depends on the specific application needs, focusing on either shared resource efficiency or independent node scalability

3. **(a) What is meant by the Open system? Briefly describe the different layers, interfaces, and protocols of distributed systems. Why the TCP/IP protocol suite is not suitable for distributed systems? [2.5 Marks]**

Scheme of evaluation: [0.5 +1.0 +1. 0] marks
Defining Open system: 0.5 mark
Description of Layers: 1.0 mark
Analysis about TCP/IP : 1.0 mark

Answer:

Open systems is defined as a computing and communication system in which the systems can able to communicate among themselves by abiding the standards and protocol irrespective of different architectures, operating systems, manufactures, networks.

The different layers are:

Physical Layer: This is the lowest layer, dealing with the actual physical connection between devices, including cables, switches, and electrical signals.

Data Link Layer: Responsible for the physical transmission of data over the network medium. It handles error detection, frame synchronization, and access to the physical medium.

Network Layer: Manages routing and forwarding of data packets across the network. This layer deals with addressing, packet switching, and network topology.

Transport Layer: Responsible for end-to-end communication between distributed systems, ensuring reliable data transfer. Protocols like TCP and UDP operate at this layer.

Application Layer: This is where user applications reside. It is the combination of three layers such as Session layer, Presentation layer and Application layer of ISO/OSI reference model.

It includes software that performs specific tasks like web browsing or file sharing. Applications communicate with each other through APIs.

TCP/IP may not be suitable for Distributed systems because of the following reasons.

- ✓ Connection oriented may not be suitable for distributed system
- ✓ Less suitable for large dynamic distributed system
- ✓ Communication on-the-fly required for distributed system, may not be supported by TCP/IP
- ✓ Communication pattern of the distributed system requires publish/subscribe, message queues, which does not supported by TCP/IP

(b) How to deal with Orphans, while a client crashes during RPC? [2.5 Marks]

Answer: When a client process crashes, then the orphan processes can be handled by four different methods. The methods are as below:

Scheme of evaluation: [1. 0 +0.5 + 0.5+0.5] marks

Method #1: Before a client stub sends an RPC message, it makes a log entry telling what it is about to do. The log is kept on disk or some other medium that survives crashes. After a reboot, the log is checked and the orphan is explicitly killed off. This solution is called extermination.

The disadvantage of this scheme is the horrendous expense of writing a disk record for every RPC. Furthermore, it may not even work, since orphans themselves may do RPCs, thus creating grand orphans or further descendants that are impossible to locate. Finally, the network may be partitioned, due to a failed gateway, making it impossible to kill them, even if they can be located.

Method #2: It is called reincarnation, all these problems can be solved without the need to write disk records. The way it works is to divide time up into sequentially numbered epochs. When a client reboots, it broadcasts a message to all machines declaring the start of a new epoch. When such a broadcast comes in, all remote computations are killed. Of course, if the network is partitioned, some orphans may survive. However, when they report back, their replies will contain an obsolete epoch number, making them easy to detect.

Method #3: It is a variant on this idea, but less Draconian. It is called gentle reincarnation. When an epoch broadcast comes in, each machine checks to see if it has any remote computations, and if so, tries to locate their owner. Only if the owner cannot be found is the computation killed.

Method #4: It is called as expiration, in which each RPC is given a standard amount of time, T, to do the job. If it cannot finish, it must explicitly ask for another quantum, which is a nuisance. On the other hand, if after a crash the server waits a time T before rebooting, all orphans are sure to be gone. The problem to be solved here is choosing a reasonable value of T in the face of RPCs with wildly differing requirements.

4. **(a) With a suitable block diagram, explain the unbuffered primitive in detail. What are the problems that occur when the client calls send() primitive before the server calls receive() primitive in an unbuffered message passing mechanism? Explain some strategies to handle these problems [2.5 Marks]**

Scheme of evaluation: [1.0 +0.5+ 1. 0] marks
Unbuffered primitive explanation : 1.0 mark
Block diagram: 0.5 mark
Problems and solutions : 1.0 mark

Answer:

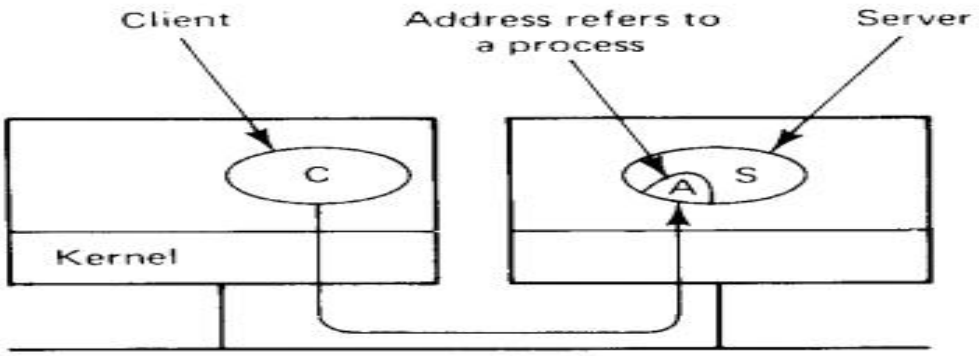
Unbuffered primitives involve direct communication without any intermediate storage.

In these primitives, the sender and receiver need to be synchronized for the communication to take place.

A call to the primitive *receive(addr, &m)* tells the kernel of the machine on which it is running that the calling process is listening to the address *addr* and is prepared to receive one message sent to that address.

A single message buffer, pointed to by m, is provided to hold the incoming message.

When the message comes in, the receiving kernel copies it to the buffer and unblocks the receiving process, as shown by the figure shown below.



The problems that occurs when the client calls send primitive before the server calls receive primitive in unbuffered message passing mechanism are:

How does the server's kernel know which of its processes is using the address in the newly arrived message?

How does the server's kernel know where to copy the message?

To avoid such problems, it's crucial to ensure that the receive primitive is called in a timely manner. Some strategies to handle this include:

Pre-emptive Design: Design the system so that the receive is always invoked before or concurrently with send to avoid blocking.

Timeouts and Error Handling: Implement timeouts or error handling mechanisms to manage situations where a send operation might block indefinitely.

Buffered Communication: Use buffered message passing where messages are stored in a buffer temporarily, allowing the sender and receiver to operate asynchronously and reducing the risk of blocking.

(b) Discuss the problems and solutions of Cristian's Algorithm used for clock synchronization in distributed systems. [2.5 Marks]

Scheme of evaluation: [1.0 +1.5] marks
Problems of Cristian's algorithm : 1.0 mark
Solutions of Cristian's algorithm:1.5 mark

Answer:

Cristian's Algorithm is used for clock synchronization in distributed systems. The issues in this algorithm are as below:

#1. The round-trip-time during message exchange can vary during communication which leads to inaccurate calculation and consequences drifting of synchronization time.

#2. Due to single point failure, the relying nodes may not be synchronized with the time server.

#3. The mechanical/hardware difference of the clocks make them drift.

#4. The algorithm does not have built-in mechanisms for verifying the authenticity of the time messages, which may lead to chaos in time-sensitive applications

#5. The algorithm is not suitable for large number of systems due to latency

Solutions for the issues of Cristian's Algorithm are as below:

#1. The RTT measurement should be improved.

#2. Multiple Server may be employed to avoid one-point failure of the time-server.

#3. Periodic resynchronization would maintain consistency across distributed system

#4. Some authenticate protocol should be integrated to ensure the authenticity fo the source of the message.

#5. Some algorithm should be integrated to measure the clock drift so as to adjust the clock proactively.

5. **(a) How a server can manage multiple client requests simultaneously in a distributed System? Analyze the the major techniques used with their pros and cons. [2.5 Marks]**

Scheme of evaluation: [1.5 +1.0] marks
Names of the mechanisms : 0.5+0.5+0.5=1.5 mark
Advantages and disadvantages:1.0 mark

Answer:

A server can manage multiple client requests simultaneously in distributed system by using the following mechanisms. Each request need to be uniquely identified by its id. The different mechanisms with their merits and demerits are as below:

#1. Multi-threading: Each client request is handled by a separate thread within the server process.
Advantages: Allows simultaneous processing of multiple requests. Provides isolation between requests.

Disadvantages: Can lead to increased memory usage and context-switching overhead.

#2. Asynchronous I/O: The server uses non-blocking I/O operations and event-driven programming to handle multiple requests.
Advantages: Efficient resource use and scalability, as the server can handle many requests with a small number of threads.
Disadvantages: More complex to implement and manage compared to multi-threading.

#3. Process-based Concurrency: Each client request is handled by a separate server process.
Advantages: Provides strong isolation between requests and fault tolerance.
Disadvantages: Higher memory and process management overhead.

(b) Suppose in a centralized approach to mutual exclusion in a distributed system, the coordinator crashes. Does this always bring the system down? If not, under what circumstances does this happen? [2.5 Marks]

Scheme of evaluation: [0.5 + 1.0 +1.0] marks
Assertion about the statement : 0.5 mark
Impact after the crash:1.0 mark
Circumstance citation: 1.0 mark

Answer:

In a centralized approach to mutual exclusion in a distributed system, a single coordinator is responsible for granting access to the critical section. If this coordinator crashes, it doesn't necessarily bring the entire system down, but it does lead to significant issues.

Impact of Coordinator crash:

i. ***Blocking of requests:*** If the coordinator crashes, any process that requests access to the critical section will be blocked indefinitely because there is no one to grant or deny access. This could lead to a system halt as processes wait indefinitely.

ii. ***System Down:*** The system might effectively be down in terms of its ability to perform critical operations that require mutual exclusion. However, other parts of the system might continue functioning if they don't require mutual exclusion.

Some of the circumstances where the system doesn't go down:

i. ***No pending requests:*** If there are no current requests for the critical section when the coordinator crashes, the system may continue operating normally until a process requests access.

ii. ***Redundancy or Backup Coordinator:*** If the system has mechanisms to detect the crash and Select a new coordinator or has a backup coordinator ready to take over, the impact can be minimized.

---: End of the Evaluation Scheme : ---