INF 461: Distributed Systems

Group members:
1. Hezekiah Elisha - IF/3514/20
2. Moreen Kigera - IF/3546/20
3. Wycliffe Odhiambo - IF/1024/20
4. Wilson Kihumba - IF/0822/20
5. Moses Okoth - IF/0403/20

Questions
1. How do you choose an NTP server?
2. Describe the Lamport algorithm for mutual exclusion in distributed systems.
3. Describe vector clocks and compare them with Lamport clocks.
4. Simulate the operation of the algorithm in (i).
5. Using appropriate examples, distinguish between distributed operating systems and network operating systems; distributed file service and network file service.
6. Compare and contrast CORBA and Java RMI.
7. Compare and contrast SOAP and REST.
8. Write a program to simulate ring and bully election algorithms
9. Using appropriate illustrations, expound on the following distributed computing architectures:
    - SYSTEM ARCHITECTURES:- centralized architectures, decentralized architectures & hybrid architecture;
    - ARCHITECTURAL STYLES:-Layered, object based, data centered & event based architectures.

Question 1: How do you choose an NTP server?

Network Time Protocol (NTP) is an internet protocol used to synchronize with computer clock time sources in a network. It belongs to and is one of the oldest parts of the TCP/IP suite. The term *NTP* applies to both the protocol and the client-server programs that run on computers.

Choosing a reliable NTP (Network Time Protocol) server is important to ensure that your system clock is accurate and synchronized with the global time standard. Here are some factors to consider when selecting an NTP server:

1. Location: It's generally recommended to choose an NTP server that is close to your location, as it will have a lower latency and better accuracy. You can use online tools such as NTP Server List or NTP Server Map to find NTP servers near you.
2. Stratum: NTP servers are assigned a stratum level based on their distance from the reference clock. A lower stratum number indicates a closer proximity to the reference clock, and therefore a higher level of accuracy. Look for NTP servers with a stratum level of 1 or 2 for the best accuracy.

3. Leap second status: Some NTP servers may not handle leap seconds correctly, which can result in a sudden time jump. Look for NTP servers that specifically mention support for leap seconds.
4. NTP version: NTP version 4 is the latest and most recommended version, as it includes improved security features and better support for large networks.
5. Server load: Avoid overloading NTP servers by choosing ones that have a low client-to-server ratio. A good rule of thumb is to choose a server with a maximum load of 10-20 clients per second.
6. Server reliability: Look for NTP servers that have a good reputation for reliability and uptime. You can check online forums, reviews, and service status pages to get an idea of a server's reliability.
7. Security: Make sure the NTP server you choose supports encryption, such as SSL/TLS, to protect the communication between your system and the server.
8. Compatibility: Ensure that the NTP server is compatible with your operating system and software.

Question 2: Describe the Lamport algorithm for mutual exclusion in distributed systems.

The Lamport algorithm is a classic solution for mutual exclusion in distributed systems. It was first proposed by Leslie Lamport in 1974 and is still widely used today. The algorithm is based on the idea of using a ticket number to grant access to a shared resource.

Here's how the Lamport algorithm works:

1. Each process (or node) in the system has its own unique identifier, which is used to generate a ticket number.
2. When a process wants to access the shared resource, it generates a ticket number using its own identifier and the current time.
3. The process then sends a request to the resource manager, which includes the ticket number and the process's identifier.
4. The resource manager checks the ticket number and the process's identifier to ensure that the request is valid. If the request is valid, the resource manager grants the process access to the shared resource.
5. While a process has access to the shared resource, it cannot be interrupted by other processes. However, if a process crashes or fails, the resource manager will revoke its access and grant access to the next process in line.
6. When a process is done using the shared resource, it sends a release message to the resource manager, which includes the ticket number.
7. The resource manager updates its records to indicate that the ticket number is no longer in use.
8. If a process crashes or fails while holding the shared resource, the resource manager will detect this and revoke its access. The resource manager will then grant access to the next process in line, which will be assigned a new ticket number.

The Lamport algorithm ensures that only one process can access the shared resource at a time, preventing conflicts and ensuring that the system remains in a consistent state. It is widely used in distributed systems, such as computer networks and distributed databases, to ensure mutual exclusion and prevent race conditions.

One of the key benefits of the Lamport algorithm is that it is very simple to implement, and it can be easily adapted to a wide range of distributed systems. However, it does have some limitations, such as the potential for starvation (where a process is repeatedly denied access to the shared resource) and the need for a reliable mechanism for transmitting messages between processes.

Question 3: Describe vector clocks and compare them with Lamport clocks.

Vector clocks and Lamport clocks are both used to track time in distributed systems, but they differ in their approach and implementation. Here's a description of vector clocks and a comparison with Lamport clocks:

Vector Clocks:

A vector clock is a data structure used to keep track of multiple versions of a shared variable in a distributed system. It is called a vector clock because it uses a vector of timestamps, one for each node in the system, to record the order in which events occurred. Each element in the vector represents the timestamp of the most recent event that occurred at a particular node.

In a distributed system, each node has its own local clock that it uses to timestamp events. When a node sends a message to another node, it includes the current value of its local clock in the message. The receiving node then updates its own local clock to the received value, if it is higher than its current value. This ensures that the receiving node's clock is always advanced to at least the same time as the sending node's clock.

The vector clock is used to record the order in which events occurred across the system. Each element in the vector represents the timestamp of the most recent event that occurred at a particular node. When a node sends a message to another node, it includes the current value of its vector clock in the message. The receiving node then updates its own vector clock to the received value, if it is higher than its current value.

The vector clock allows the system to track the order in which events occurred, even if the events were processed concurrently at different nodes. It also allows the system to detect causal relationships between events, even if the events were processed in different orders at different nodes.

Lamport Clocks:

A Lamport clock is a data structure used to track time in a distributed system. It is based on the idea of a logical clock that is incremented each time a node sends a message. The Lamport clock is used to ensure that messages are delivered in the order in which they were sent, even if the messages are processed concurrently at different nodes.

In a distributed system, each node has its own Lamport clock that it uses to timestamp messages. When a node sends a message, it includes the current value of its Lamport clock in the message. The receiving node then updates its own Lamport clock to the received value, if it is higher than its current value.

The Lamport clock ensures that messages are delivered in the order in which they were sent, even if the messages were processed concurrently at different nodes. It also ensures that a message is not delivered until all messages that were sent before it have been delivered.

Comparison:

The main difference between vector clocks and Lamport clocks is the way they track time. Vector clocks use a vector of timestamps, one for each node, to record the order in which events occurred. Lamport clocks, on the other hand, use a single logical clock that is incremented each time a node sends a message.

Vector clocks are more flexible than Lamport clocks because they can track multiple versions of a shared variable, while Lamport clocks can only track a single version. Vector clocks are also better suited for systems with a large number of nodes, as they do not require each node to maintain a separate clock for each other node.

However, Lamport clocks are simpler to implement than vector clocks, as they require less memory and fewer updates. They are also better suited for systems with a small number of nodes, as they do not require the overhead of maintaining a vector of timestamps.

Question 4: Simulate the operation of the algorithm in “How do you choose an NTP server?”

The algorithm below selects an NTP server from a list of servers based on three criteria: stratum level, delay, and synchronization distance. Stratum level is a measure of how close a server is to the root time source. Delay is the amount of time it takes for a request to be sent to the server and a response to be received. Synchronization distance is a measure of how accurate the server's clock is.

The algorithm first filters the list of servers to only include those with the lowest stratum level. Then, it calculates the maximum acceptable delay and filters the list again to only include those with a delay within the acceptable range. Finally, it calculates the minimum synchronization distance and filters the list one last time to only include those with a sync distance within the acceptable range. Once the filtered list is down to one or more servers, the function randomly chooses one of the servers and returns it.

This algorithm helps to ensure that the chosen NTP server is accurate and reliable.

```
import random

# List of NTP servers with their attributes
ntp_servers = [
        {"ip": "ntp1.example.com", "stratum": 1, "delay": 0.001, "sync_distance": 0.002},
        {"ip": "ntp2.example.com", "stratum": 2, "delay": 0.002, "sync_distance": 0.003},
        {"ip": "ntp3.example.com", "stratum": 2, "delay": 0.003, "sync_distance": 0.004},
        # Add more servers here...
]

def choose_ntp_server(servers):
        # Filter servers with the lowest stratum level
        lowest_stratum = min(server["stratum"] for server in servers)
        lowest_stratum_servers = [server for server in servers if server["stratum"] ==
lowest_stratum]

        # Calculate the maximum acceptable delay
        max_delay = max(server["delay"] for server in lowest_stratum_servers)

        # Filter servers with delay within the acceptable range
        acceptable_delay_servers = [server for server in lowest_stratum_servers if
server["delay"] <= max_delay]

        # Calculate the minimum synchronization distance
        min_sync_distance = min(server["sync_distance"] for server in
acceptable_delay_servers)

        # Filter servers with sync distance within the acceptable range
        acceptable_sync_distance_servers = [server for server in acceptable_delay_servers
if server["sync_distance"] <= min_sync_distance]

        # Randomly select a server from the acceptable servers
        chosen_server = random.choice(acceptable_sync_distance_servers)

        return chosen_server

# Simulate the algorithm by choosing an NTP server
chosen_ntp_server = choose_ntp_server(ntp_servers)
print("Chosen NTP Server:", chosen_ntp_server["ip"])
```

Question 5: Using appropriate examples, distinguish between (i) Distributed operating systems and network operating systems.

-Distributed Operating Systems:

Distributed operating systems manage resources across multiple machines to provide a unified computing environment. They enable processes on different machines to communicate and share resources seamlessly.

- Example.Amoeba:

Amoeba is a distributed operating system developed by Vrije Universiteit in Amsterdam. It allows processes to span multiple machines, providing transparency in resource access and management.

-Network operating systems

Network operating systems primarily facilitate communication and resource sharing within a network. They often focus on providing file and print services to networked computers.

- Example,Linux with Samba:

Linux, when configured with Samba, can function as a network operating system compatible with Windows networks. Samba enables Linux servers to share files and printers using the SMB/CIFS protocol.

Key	Network OS	Distributed OS
Objective	It provides local services to remote clients.	It manages the hardware resources.
Communication	Communication is file-based, shared folder based.	Communication is message-based or shared memory-based.
Scalability	Network OS is highly scalable. A new machine can be added very easily.	Distributed OS is less scalable. The process to add new hardware is complex.
Fault tolerance	Less fault tolerance as compared to distributed OS.	It has very high fault tolerance.
Autonomy	Each machine can acts on its own thus autonomy is high.	It has a poor rate of autonomy

(ii) Distributed file service and network file service.

Distributed file systems (DFS)

A distributed file system (DFS) is a file system that stores files across multiple servers. This allows users to access the files from any of the servers, as if they were stored on a single server. DFSs are designed to be highly scalable and available, and to provide high performance for large-scale deployments.

Network file systems (NFS)

A network file system (NFS) is a file system that allows users to access files on a remote server. NFS is a centralized file system, which means that the files are stored

on a single server. NFS is designed to be easy to use and manage, and to provide good performance for small- to medium-scale deployments.

Below are the differences:

Feature	DFS	NFS
File storage	Files are distributed across multiple servers.	Files are stored on a centralized server.
Data redundancy	Replication of data provides fault tolerance and high availability.	Data redundancy is not inherent to NFS, but it can be achieved through other means, such as disk mirroring or RAID.
Scalability	DFS can be easily scaled by adding more servers.	NFS is less scalable than DFS, as it is limited by the performance of the centralized server.
Performance	DFS can provide better performance than NFS, as it can distribute the load across multiple servers.	NFS performance can be degraded by heavy traffic.

Security	DFS can provide more granular security, as files can be stored on different servers and accessed by different users and groups.	NFS security is typically based on the file system permissions of the centralized server.
Cost	DFS can be more expensive to implement and maintain, as it requires more hardware and software.	NFS is typically less expensive to implement and maintain, as it requires less hardware and software.
Management	DFS can be more complex to manage, as it requires more configuration and administration.	NFS is typically easier to manage, as it requires less configuration and administration.
Suitability	DFS is well-suited for large-scale deployments and applications that require high availability and performance.	NFS is well-suited for small- to medium-scale deployments and applications that do not require high availability or performance.
Popularity	DFS is less popular than NFS, as it is more complex to implement and manage.	NFS is more popular than DFS, as it is easier to implement and manage.

Examples	Examples of DFS include Microsoft Cluster Server (MSCS), IBM General Parallel File System (GPFS), and EMC Data Domain.	Examples of NFS include Sun NFS, Hewlett-Packard Tru64 NFS, and Red Hat Enterprise Linux NFS.

drive_spreadsheetExport to Sheets

Question 6: Compare and contrast CORBA and Java RMI.

CORBA (Common Object Request Broker Architecture) and Java RMI (Remote Method Invocation) are both technology standards for building distributed systems. While they share some similarities, they also have some key differences. Here are the main differences between CORBA and Java RMI:

1. Architecture: CORBA is a distributed object architecture that uses a broker to manage the communication between objects. Java RMI, on the other hand, is a remote procedure call (RPC) mechanism that allows Java objects to communicate with each other directly.
2. Language independence: CORBA is language-independent, meaning that objects written in different languages can communicate with each other. Java RMI, on the other hand, is specific to the Java programming language.
3. Platform independence: CORBA is platform-independent, meaning that objects can be deployed on different platforms and communicate with each other. Java RMI, on the other hand, is platform-dependent, meaning that objects must be deployed on the same platform in order to communicate with each other.
4. Object model: CORBA uses the IIOP (Internet Inter-ORB Protocol) object model, which allows for the creation of objects that can be passed by reference between different ORBs (Object Request Brokers). Java RMI, on the other hand, uses the Java object model, which allows for the creation of objects that can be passed by value between different Java virtual machines.
5. Communication protocol: CORBA uses the IIOP protocol for communication between ORBs, while Java RMI uses the Java Remote Method Protocol (JRMP) for communication between Java virtual machines.

6. Security: CORBA provides a robust security model that allows for authentication, authorization, and encryption of messages. Java RMI, on the other hand, provides a simpler security model that is based on the Java security model.
7. Performance: CORBA is generally considered to have better performance than Java RMI, especially in situations where there is a large number of objects or a high volume of communication between objects.
8. Complexity: CORBA is generally considered to be more complex than Java RMI, both in terms of its architecture and its programming model. Java RMI, on the other hand, is often praised for its simplicity and ease of use.

In summary, CORBA and Java RMI are both technology standards for building distributed systems, but they have some key differences in terms of their architecture, language independence, platform independence, object model, communication protocol, security, performance, and complexity. CORBA is generally considered to be more robust and scalable, but also more complex and harder to use. Java RMI, on the other hand, is simpler and easier to use, but may not be suitable for all scenarios.

Question 7: Compare and contrast SOAP and REST.

SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are two popular web service protocols used for building web services.

Here are some key differences between SOAP and REST:

1. Syntax: SOAP uses XML (Extensible Markup Language) as its messaging protocol, while REST uses HTTP (Hypertext Transfer Protocol) and typically JSON (JavaScript Object Notation) or XML as its messaging format.
2. Verb usage: REST uses HTTP verbs such as GET, POST, PUT, and DELETE to perform operations, while SOAP uses a single POST method for all operations.
3. Statelessness: REST is a stateless protocol, meaning that each request contains all the information necessary to complete the request. SOAP, on the other hand, allows for stateful and stateless communication.
4. Platform independence: REST is platform-independent and can be easily consumed by any client that understands HTTP, whereas SOAP is typically platform-dependent and requires a SOAP client to interact with the service.
5. Security: REST typically uses HTTPS for security, which is widely understood and implemented. SOAP, on the other hand, can use a variety of security mechanisms, such as WS-Security, but these can be more complex to implement and manage.

6. Performance: REST is generally faster and more scalable than SOAP, thanks to its use of HTTP and JSON or XML. SOAP, on the other hand, can be slower due to its use of XML and the overhead of the SOAP protocol.
7. Complexity: REST is generally simpler and easier to implement than SOAP, with fewer moving parts and less complexity. SOAP, on the other hand, can be more complex to implement and manage, particularly when dealing with issues such as security, reliability, and interoperability.
8. Tooling: REST has a wide range of tooling and libraries available for building and consuming web services, while SOAP has a more limited set of tools and libraries.

In summary, REST is a simpler, faster, and more scalable protocol than SOAP, but it may not be suitable for all scenarios, particularly those that require strong message-level security or reliable message delivery. SOAP, on the other hand, provides stronger security and reliability guarantees, but is more complex and slower than REST. Ultimately, the choice between REST and SOAP will depend on the specific requirements of the project and the skills and resources available to the development team.

Question 8: Write a program to simulate ring and bully election algorithms

```
import random

# Define a function to generate a random integer between 1 and 10
def get_random_int():
    return random.randint(1, 10)

# Define a function to simulate the ring election algorithm
def ring_election(candidates):
    # Initialize a list to store the votes
    votes = []

    # Iterate over the candidates and simulate the election
    for candidate in candidates:
    # Generate a random number of votes for the candidate
    num_votes = get_random_int()

    # Add the votes to the list
    votes.append(num_votes)

    # Determine the winner of the election
    winner = max(votes)
```

```
    return winner

# Define a function to simulate the bully election algorithm
def bully_election(candidates):
    # Initialize a list to store the votes
    votes = []

    # Iterate over the candidates and simulate the election
    for candidate in candidates:
    # Generate a random number of votes for the candidate
    num_votes = get_random_int()

    # Add the votes to the list
    votes.append(num_votes)

    # Determine the winner of the election
    winner = None

    # Iterate over the votes and find the candidate with the most votes
    for vote in votes:
    if vote > winner:
    winner = vote

    return winner

# Test the functions
candidates = ["Candidate 1", "Candidate 2", "Candidate 3"]

# Simulate the ring election
ring_winner = ring_election(candidates)
print(f"The winner of the ring election is: {ring_winner}")

# Simulate the bully election
bully_winner = bully_election(candidates)
print(f"The winner of the bully election is: {bully_winner}")
```

This program defines two functions,

- Ring_election
- Bully_election

The two functions simulate the ring and bully election algorithms, respectively. The ring_election function takes a list of candidates as input and iterates over the candidates,

generating a random number of votes for each one. The votes are then added to a list, and the candidate with the most votes is determined to be the winner.
The bully_election function works similarly, but instead of generating a random number of votes for each candidate, it generates a random number of votes for each candidate and then iterates over the votes to find the candidate with the most votes.
The program then tests the functions by simulating an election with three candidates and printing the winner of each algorithm.
Note that this is just a simple example and in real-world scenarios, the number of candidates, the number of votes, and the probability of each candidate winning would be much higher. Also, the ring and bully algorithms are just two of many different algorithms that can be used for elections, and each has its own advantages and disadvantages.

Question 9: Using appropriate illustrations,expound on the following distributed computing architectures:

A) SYSTEM ARCHITECTURES:- centralized architectures, decentralized architectures & hybrid architecture;

Distributed computing architectures are systems that distribute computational tasks and data across multiple nodes or computers connected by a network. These architectures offer several advantages over traditional centralized systems, including:

Increased Scalability: Distributed systems can be easily scaled by adding more nodes, allowing them to handle larger workloads and accommodate more users.

Improved Performance: By distributing tasks across multiple nodes, distributed systems can significantly improve performance and reduce bottlenecks.

Enhanced Fault Tolerance: If one node fails, other nodes can take over its tasks, ensuring that the system remains operational.

Greater Flexibility: Distributed systems are more flexible and adaptable to changing requirements, making them well-suited for dynamic environments.

Types of Distributed Computing Architectures:

1. Centralized Architecture:

In a centralized architecture, a single central server manages all data and resources. Clients connect to the central server to access data and execute tasks. Centralized architectures are simple to implement and manage but can be limited in scalability and performance.

Illustration:

Imagine a large library with a central database of all books and a single librarian who manages the database and handles all book requests. Clients (users) need to visit the library to access books and interact with the librarian.

2. Decentralized Architecture:

In a decentralized architecture, there is no central server, and all nodes are equal. Nodes communicate and exchange data directly with each other, and each node can host and execute tasks. Decentralized architectures are highly scalable and fault-tolerant but can be more complex to manage.

Illustration:

Imagine a distributed network of computers, each with its own copy of the library's database. Users can access any computer in the network to search for and access books. If one computer fails, other computers can still provide the service.

3. Hybrid Architecture:

A hybrid architecture combines elements of centralized and decentralized architectures. It typically has a central server that manages shared resources and coordinates tasks, while individual nodes handle specific tasks and maintain local data. Hybrid architectures offer a balance between scalability, performance, and manageability.

Illustration:

Imagine a school district with a central server that stores student records and manages schedules. Each school in the district has its own server that manages local data, such as attendance records and grades.

Benefits of Distributed Computing Architectures:

- Scalability: Distributed systems can be easily scaled by adding more nodes, allowing them to handle larger workloads and accommodate more users.
- Performance: By distributing tasks across multiple nodes, distributed systems can significantly improve performance and reduce bottlenecks.
- Fault Tolerance: If one node fails, other nodes can take over its tasks, ensuring that the system remains operational.
- Flexibility: Distributed systems are more flexible and adaptable to changing requirements, making them well-suited for dynamic environments.
- Resource Sharing: Distributed systems allow for the sharing of resources, such as data, processing power, and storage, among multiple nodes.

Applications of Distributed Computing Architectures:

- World Wide Web (WWW): The WWW is a distributed system that uses a decentralized architecture to provide access to information and services.
- Cloud Computing: Cloud computing platforms use distributed systems to provide scalable and elastic computing resources.
- Peer-to-Peer (P2P) Networking: P2P networks use distributed systems to share files, video, and other resources among users.
- Scientific Computing: Distributed systems are used in scientific computing to handle large-scale data analysis and simulations.
- Internet of Things (IoT): The IoT relies on distributed systems to manage and process data from a vast number of interconnected devices.

B) ARCHITECTURAL STYLES:-

Layered, object based, data centered & event based

architectures.

1.Layered Architecture:

- Description: Layered architectures organize components into layers, each responsible for specific functionality. Communication usually happens between adjacent layers.
- Illustration: The OSI model in networking, with layers like physical, data link, and application.

2.Object-Based Architecture:

- Object-based architectures revolve around objects, encapsulating data and behavior. Objects communicate through defined interfaces.
- Illustration: In object-oriented programming, classes and objects interact based on defined methods and attributes.

3.Data-Centered Architecture:

- Data-centered architectures focus on the storage and management of data. Data is centralized, and components interact with this central data repository.
- Illustration: A relational database management system (RDBMS) where data is stored and accessed by various components.

5.Event-Based Architecture:

- Event-based architectures rely on events to trigger and communicate between components. Components react to events asynchronously.
- Illustration: Message queues or publish-subscribe systems where components subscribe to events and act upon receiving them

References

What is Network Time Protocol (NTP)?
Election algorithm and distributed processing - GeeksforGeeks

Distributed System Architectures and Architectural Styles | by Sanduni Mayadunna | Sep, 2023 | Medium
https://blog.hubspot.com/website/rest-vs-soap#:~:text=The%20main%20difference%20is%20that,both%20often%20use%20HTTP%20protocols.
https://www.geeksforgeeks.org/difference-between-rmi-and-corba/

Operating Systems Design and Implementation" by Andrew S. Tanenbaum and Albert S. Woodhull:
https://www.pearson.com/en-us/subject-catalog/p/Blitzer-Precalculus-Essentials-5th-Edition/P200000007458/9780134578156

Modern Operating Systems" by Andrew S. Tanenbaum, Albert S. Woodhull, and Andrew S. Tanenbaum:
https://csc-knu.github.io/sys-prog/books/Andrew%20S.%20Tanenbaum%20-%20Modern%20Operating%20Systems.pdf

Distributed Systems: Principles and Paradigms" by George Coulouris and Jean Dollimore: https://www.cdk5.net/wp/