Practical 1: Client/Server Process Communication using RMI

Overview:

This practical focuses on implementing communication between two separate Java programs (a client and a server) running potentially on different machines. It utilizes **Remote Method Invocation (RMI)**, a Java API that allows a Java object running in one Java Virtual Machine (JVM) to invoke methods on a remote Java object running in another JVM.

Think of it like this: Imagine you have an application running on your computer (the client) that needs to use a service offered by another application running on a different computer (the server). RMI provides the mechanism for your client application to call functions (methods) in the server application as if they were running locally.

The practical likely involves the following steps:

1. **Defining a Remote Interface:** You'll create an interface in Java that declares the methods the client can call on the server. This acts as a contract between the client and the server.
2. **Implementing the Remote Interface on the Server:** The server-side program will have a class that implements the remote interface. This class contains the actual logic for the methods that can be called remotely.
3. **Registering the Remote Object:** The server needs to make its remote object accessible. This is done using the **RMI registry**, a special naming service that runs on the server machine. The server registers its remote object with a specific name in the registry.
4. **Looking Up the Remote Object on the Client:** The client program contacts the RMI registry on the server, looks up the remote object by its registered name, and obtains a reference to it (called a "stub").
5. **Invoking Remote Methods:** The client can then use the stub as if it were a local object and call the methods defined in the remote interface. The RMI infrastructure handles the underlying network communication, marshaling (packaging) the arguments, sending them to the server, unmarshaling them on the server, executing the method, and then sending back the results in a similar way.
6. **Multi-threading:** The problem statement mentions "multi-threaded client/server." This means the server will likely be designed to handle multiple client requests concurrently using threads. When a client connects, the server might create a new thread to handle that specific client's requests, allowing it to serve other clients simultaneously.

Potential Viva Questions and Answers:

1. **What is RMI?**
    - **Answer:** RMI (Remote Method Invocation) is a Java API that allows a Java object running in one JVM to invoke methods on a remote Java object running in another JVM. It simplifies the development of distributed Java applications.
2. **What is the purpose of the RMI registry?**
    - **Answer:** The RMI registry is a naming service that allows server objects to register themselves with a name and allows clients to look up these remote objects by their name. It acts as a central directory for remote objects.
3. **What are the key components involved in RMI?**
    - **Answer:** The key components are:
        - **Remote Interface:** Defines the methods that can be invoked remotely.
        - **Remote Object (Server-side):** The object that implements the remote interface and whose methods can be called remotely.
        - **Stub (Client-side):** A client-side proxy for the remote object. It has the same methods as the remote interface and handles the communication with the server.
        - **Skeleton (Server-side - less explicitly used now but conceptually important):** A server-side object that receives remote calls, unmarshals the arguments, and dispatches the call to the actual remote object.
        - **RMI Registry:** The naming service.
4. **What is the role of serialization in RMI?**
    - **Answer:** Serialization is the process of converting an object's state into a byte stream. In RMI, arguments and return values of remote method calls need to be serialized on the sender's side and deserialized (converted back to objects) on the receiver's side for transmission over the network.
5. **What is multi-threading in the context of a server? Why is it important?**
    - **Answer:** Multi-threading in a server means the server can handle multiple client requests concurrently by creating a separate thread for each client connection or request. This is important because it allows the server to

be more responsive and efficient, as it doesn't have to wait for one client's request to finish before serving another.

6. **How does a client find a remote object in RMI?**
    - **Answer:** The client contacts the RMI registry running on the server's machine, using the server's IP address and the registry's port number (usually 1099). It then looks up the remote object by its registered name. The registry returns a stub (proxy) of the remote object to the client.
7. **What are some potential issues or challenges in RMI?**
    - **Answer:** Some potential issues include network connectivity problems, security considerations (as data is transmitted over the network), dealing with exceptions during remote method calls, and the need to ensure that both the client and server have the necessary classes.
8. **What is a socket, and how does RMI relate to sockets?**
    - **Answer:** A socket is an endpoint for network communication. It's a combination of an IP address and a port number. RMI uses sockets under the hood for the actual network communication between the client and the server. While you don't directly work with sockets in RMI, the RMI framework handles the socket creation and management for you.

Practical 4: Berkeley Algorithm

Overview:

This practical deals with **clock synchronization** in distributed systems. In a distributed system, each computer has its own clock, and these clocks can drift apart over time, leading to inconsistencies. The Berkeley Algorithm is designed to synchronize these clocks to maintain a consistent time across the system.

Here's how it generally works:

1. **Master Selection:** One computer is designated as the "master" or coordinator.
2. **Time Collection:** Periodically, each other computer (slave) sends its current time to the master.
3. **Average Calculation:** The master collects these times and calculates the average time. This average may or may not include the master's time and might involve some adjustments to account for network delays.
4. **Time Update:** The master sends the calculated average time (or an adjustment value) back to the slaves.
5. **Clock Adjustment:** Each slave adjusts its own clock based on the information received from the master.

Key Points:

- The Berkeley Algorithm is a **centralized** algorithm, meaning it relies on a single master.
- It's designed to work even if some machines don't have accurate time sources.
- It's important to repeat the synchronization process periodically to counteract clock drift.
- Alternatives like the Network Time Protocol (NTP) often provide more accurate synchronization.

Potential Viva Questions and Answers:

1. **What is the Berkeley Algorithm?**
    - **Answer:** It's a clock synchronization algorithm for distributed systems where a master computer periodically collects the times from other computers, calculates an average, and tells the other computers how to adjust their clocks.
2. **Why is clock synchronization important in distributed systems?**

- **Answer:** To maintain consistency of events and ordering of operations across the system. Inconsistent clocks can lead to errors, especially in applications that rely on accurate timestamps (e.g., transaction processing, logging).

3. **How does the Berkeley Algorithm choose the master?**
    - **Answer:** The document mentions that the master node is chosen using an election process/leader election algorithm. It doesn't specify the exact election algorithm.
4. **What happens if the master fails?**
    - **Answer:** The document mentions fault tolerance by using backup time coordinators. If the master fails, the algorithm needs a mechanism to choose a new master, which could involve another election.
5. **What are the limitations of the Berkeley Algorithm?**
    - **Answer:** The accuracy is limited by network conditions and delays. It also relies on a single master, which can be a single point of failure.
6. **How is the average time calculated?**
    - **Answer:** The master computer calculates the average time of all the timestamps it has received.
7. **How do the slave nodes adjust their clocks?**
    - **Answer:** Each computer sets its clock to the time it receives from the master computer.
8. **What is NTP? How is it different from the Berkeley Algorithm?**
    - **Answer:** NTP (Network Time Protocol) is another clock synchronization protocol. It's generally more complex and accurate than the Berkeley Algorithm, as it considers network delays and clock drift in more detail.
9. **What are the features of Berkeley's Algorithm?**
    - **Answer:** Centralized time coordinator, clock adjustment, average calculation, fault tolerance, accuracy and scalability.

Practical 5: Implement token ring based mutual exclusion algorithm

Overview:

This practical focuses on **mutual exclusion** in distributed systems. Mutual exclusion is a crucial concept that ensures that only one process can access a shared resource at any given time, preventing data corruption and inconsistencies. The token ring algorithm is one way to achieve mutual exclusion in a distributed environment.

Here's how the token ring algorithm works:

1. **Logical Ring:** Processes in the distributed system are organized in a logical ring. This doesn't necessarily mean they are physically arranged in a ring, but rather that each process knows its predecessor and successor.
2. **Token Circulation:** A special message called a "token" circulates around the ring.
3. **Critical Section Access:** A process can only enter its critical section (the code that accesses the shared resource) if it holds the token.
4. **Token Passing:** After using the token (or if it doesn't need it), the process passes the token to its neighbor in the ring.

Key Points:

- The token ensures that only one process can be in the critical section at a time.
- The order of the processes in the ring determines the order in which they can access the critical section.
- The algorithm needs mechanisms to handle situations like token loss or process failure.

Potential Viva Questions and Answers:

1. **What is mutual exclusion?**
    - **Answer:** Mutual exclusion is a concurrency control mechanism that prevents multiple processes from accessing a shared resource simultaneously.
2. **Why is mutual exclusion important in distributed systems?**
    - **Answer:** Because distributed systems often involve shared resources, and mutual exclusion is necessary to maintain data integrity and prevent race conditions.
3. **Explain the token ring algorithm.**
    - **Answer:** It's a distributed mutual exclusion algorithm where a unique token circulates among processes arranged in a logical ring. Only the process holding the token can enter its critical section.
4. **How is the logical ring organized?**

- **Answer:** Each process knows its predecessor and successor, forming a circular sequence.

5. **How does a process get permission to enter the critical section in the token ring algorithm?**
    - **Answer:** A process can enter the critical section only if it possesses the token.
6. **What happens after a process uses the token?**
    - **Answer:** It passes the token to the next process in the ring.
7. **What are the requirements of a good mutual exclusion algorithm?**
    - **Answer:** The requirements are No Deadlock, No Starvation, Fairness and Fault Tolerance.
8. **What are some other mutual exclusion algorithms?**
    - **Answer:** The document mentions Non-token based approach (Lamport's algorithm, Ricart–Agrawala algorithm) and Quorum based approach (Maekawa's Algorithm).
9. **What are the advantages and disadvantages of the token ring algorithm?**
    - **Answer:**
        - **Advantages:** Relatively simple to implement. Provides fair access if the ring is fair.
        - **Disadvantages:** Can be inefficient if processes rarely need the critical section (token keeps circulating). A single point of failure (if a process fails, the token can be lost). Overhead of token management.

Practical 6: Bully and Ring Algorithm

Overview:

This practical deals with **leader election** in distributed systems. In many distributed systems, it's necessary to have a coordinator or leader process to manage certain tasks. If the current leader fails, an election must be held to choose a new one. The Bully and Ring algorithms are two classic algorithms for leader election.

Bully Algorithm:

- **Assumption:** Each process has a unique ID, and processes know the IDs of other processes.
- **Election Initiation:** When a process notices the coordinator is not responding, it initiates an election.
- **Election Process:** The process sends an ELECTION message to all processes with higher IDs.
    - If no one responds, it becomes the new coordinator.
    - If a higher-ID process responds, that process takes over the election.
- **Coordinator Announcement:** The new coordinator announces its victory to all other processes.
- The "biggest" process always wins, hence the name "Bully."

Ring Algorithm:

- **Assumption:** Processes are arranged in a logical ring.
- **Election Initiation:** Any process can start an election by sending an ELECTION message containing its own ID to its successor in the ring.
- **Election Process:** Each process forwards the ELECTION message to its successor, adding its own ID to the message.
- **Coordinator Selection:** When the message returns to the initiator, the process with the highest ID is chosen as the leader.
- **Coordinator Announcement:** The initiator sends a message around the ring to announce the new coordinator.

Key Points:

- Both algorithms ensure that a new leader is chosen if the current one fails.
- They have different assumptions about the system structure and communication patterns.

Potential Viva Questions and Answers:

1. **What is a leader election algorithm?**
    - **Answer:** It's an algorithm that chooses a process to act as a coordinator or leader in a distributed system.
2. **Why are leader election algorithms needed in distributed systems?**

- **Answer:** To ensure that there is a process to coordinate tasks, make decisions, or manage resources. If the current coordinator fails, a new one must be chosen to maintain system functionality.

3. **Explain the Bully Algorithm.**
    - **Answer:** It's an election algorithm where the process with the highest ID becomes the leader. Processes initiate elections when they detect a failure and assert their dominance based on their ID.
4. **Explain the Ring Algorithm.**
    - **Answer:** It's an election algorithm where processes are organized in a logical ring, and an election message is passed around the ring. The process with the highest ID is elected as the leader.
5. **What are the assumptions of the Bully Algorithm?**
    - **Answer:** Each process has a unique ID, and processes know each other's IDs.
6. **What are the assumptions of the Ring Algorithm?**
    - **Answer:** Processes are arranged in a logical ring, and communication is unidirectional.
7. **How is a new coordinator chosen in the Bully Algorithm?**
    - **Answer:** The process with the highest ID that is still functioning is chosen as the new coordinator.
8. **How is a new coordinator chosen in the Ring Algorithm?**
    - **Answer:** The process with the highest ID among the participants in the election is chosen as the new coordinator.
9. **What are the advantages and disadvantages of the Bully Algorithm?**
    - **Answer:**
        - **Advantages:** Relatively simple to understand.
        - **Disadvantages:** Can lead to unnecessary elections if a recovered process has a high ID but isn't needed as the coordinator.

10. **What are the advantages and disadvantages of the Ring Algorithm?**
    - **Answer:**
        - **Advantages:** Fairer than the Bully Algorithm in some scenarios.
        - **Disadvantages:** Can be slower than the Bully Algorithm, as messages must circulate the entire ring in the worst case.

Practical 7:- Code explained of the Java Web Application**

The provided zip file, `Web Service Calculator.zip`, likely contains a Java web application that functions as a calculator through web services. Below is a structured explanation of the components, libraries, and modules that are typically found in such applications.

1. Project Structure

- src/: Contains the source code of the application.
- lib/: Contains external libraries (JAR files) used in the application.
- -WEB-INF/: Contains configuration files and the web application deployment descriptor (web.xml).
- index.html: The main HTML file that serves as the user interface.

2. Libraries Used

- **Servlet API**: For handling HTTP requests and responses.

- **JSP (JavaServer Pages)**: For creating dynamic web content.

- **JSON Processing Library**: For handling JSON data (e.g., Jackson or Gson).

- **JUnit**: For unit testing the application.

- **Log4j**: For logging application events.

3. Modules Explanation

3.1. User Interface (UI

- **index.html:**

- Provides a simple form for users to input numbers and select operations (addition, subtraction, multiplication, division).

- Uses JavaScript for client-side validation and AJAX calls to the server.

3.2. Backend Logic

- **CalculatorServlet.java:**

- Extends `HttpServlet` to handle HTTP requests.

- Processes input from the UI, performs calculations, and returns results.

- Uses `doGet()` or `doPost()` methods to handle requests.

- **CalculatorService.java:**

- Contains the business logic for performing calculations.

- Methods include `add()`, `subtract()`, `multiply()`, and `divide()`.

- Handles exceptions (e.g., division by zero).

3.3. Configuration Files

- **web.xml:**

- Deployment descriptor that defines servlet mappings, initialization parameters, and other configuration settings.

- Maps the `CalculatorServlet` to a specific URL pattern (e.g., `/calculate`).

4. Working of the Application

1. **User Input**:

   - The user accesses `index.html` and inputs numbers and selects an operation.

2. **AJAX Call**:

   - Upon form submission, an AJAX call is made to the `CalculatorServlet` with the input data.

3. **Servlet Processing**:

   - The `CalculatorServlet` receives the request, extracts parameters, and calls the appropriate method in `CalculatorService`.

4. **Calculation**:

   - The `CalculatorService` performs the requested operation and returns the result.

5. **Response**:

   - The servlet sends the result back to the client, which updates the UI dynamically without a full page reload.

5. Conclusion

The Java web application functions as a simple calculator that utilizes servlets for backend processing and HTML/JavaScript for the frontend. The application is structured to separate concerns, with distinct modules for UI, business logic, and configuration. The use of libraries enhances functionality and maintainability.