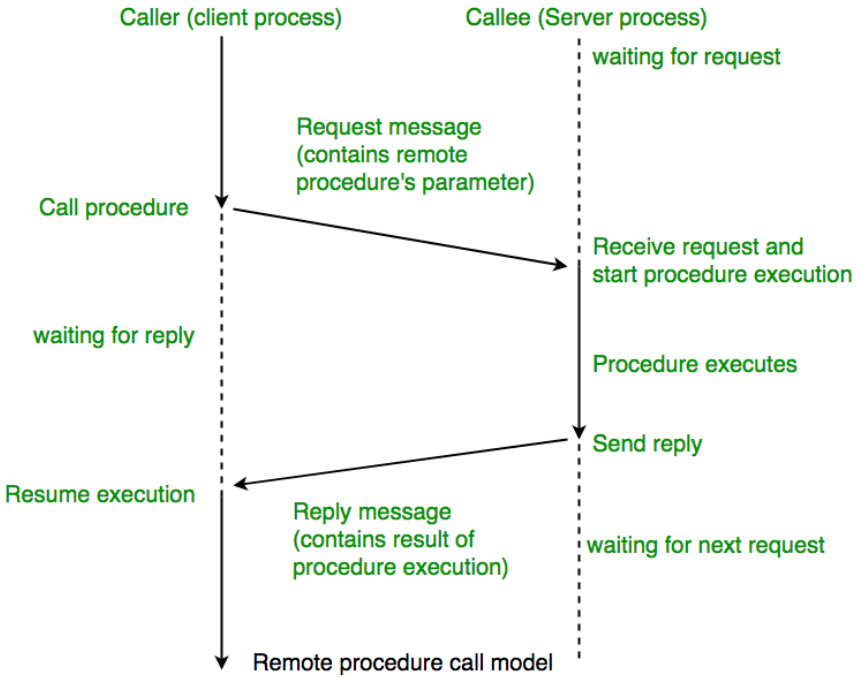
Remote Procedure Call (RPC)

- procedure need not exist in the same address space as the calling procedure



-
- Based on *client-server architecture*
- Working of RPC

common protocols used in RPC : HTTP, XML-RPC, JSON-RPC

transport layer handles data transfer and reliability between client and server during RPC calls

XML - RPC (Extensible Markup Language - RPC)
It uses XML to encode calls and HTTP as a transport mechanism, allowing interoperability across different platforms.

Features of XML - RPC :
1) XML for Data Encoding
2) HTTP for Transport
3) ability to call a procedure (function) on a remote server

`SimpleXMLRPCServer` **creates a simple XML-RPC server** that listens for incoming RPC requests on a specified address and port.

`server.serve_forever()` **starts the server**, putting it into a loop where it continuously listens for and **handles incoming XML-RPC requests**.

`logRequests=False` **prevents the server from logging each request to the console.** This can be useful if you want a cleaner output without HTTP request logs.

xmlrpc.client.ServerProxy
`ServerProxy` creates a client object

What happens if the server is not running when the client tries to connect?

- **ConnectionError**

client will raise a connection error bcz no server is present at the specified address to handle the request.

XML-RPC over HTTP is not secure. (https - secure)

Port Numbers
0-1023 : **well-known" or "privileged" ports** - reserved

ports above 1023 : **ephemeral or dynamic ports -** for custom applications or testing

CLOCK SYNCHRONIZATION

process of ensuring that all clocks across various nodes or computers in the system are set to the same time

NEED :

1. Event Ordering
2. timestamping data operations accurately
3. Timestamps - crucial for security protocols
4. efficient fault detection and recovery mechanisms

Multithreading : NTP and Lamport servers in separate threads, the code can handle requests to both servers simultaneously.

NTP - Network Time Protocol

- Oldest
- high accuracy.
- Client-Server Architecture

NTP **compares timestamps** from multiple time servers, **calculates the offset** (difference in time), and **adjusts the local clock** gradually to minimize error.

Other Protocol : **Precision Time Protocol (PTP)**

NTP Server: This server provides a timestamp in NTP format, which uses an epoch starting on January 1, 1900. When a client requests the time, the server responds with the current timestamp adjusted to the NTP epoch.

timestamp = int(time.time()) + 2208988800
converts the Unix timestamp (starting from January 1, 1970) to the NTP timestamp by adding an offset of `2208988800` seconds.

`NTP_OFFSET` converts the **NTP timestamp** (based on the 1900 epoch) to the **Unix timestamp format** (based on the 1970 epoch). This constant allows for **accurate time conversion** between the two format

\x1b : to indicate a client request

NTP Response
response = b'\x1b' + (b'\0' * 39) + timestamp.to_bytes(4, 'big') + (b'\0' * 4)

`b'\x1b' -` 1 byte - `0x1b` in hexadecimal - 27 in decimal - client req
`(b'\0' * 39)`: 39 bytes of zeros
`.to_bytes(4, 'big')` converts this integer timestamp to a **4-byte big-endian format**
(b'\0' * 4) : 4 bytes of zeros

1 byte for the **LI(2) = 00, VN(3)=100, and Mode(3)=011** information.(**Leap Indicator , Version Number and mode**)
39 bytes of zeros for unused fields.
4 bytes representing the **current timestamp** in NTP format.
4 trailing bytes of zeros to complete the 48-byte packet length required by NTP.

Real-World Examples of Clock Synchronization in Distributed Systems

1. Distributed Databases
2. Cloud Computing

LAMPORT CLOCK

logical clock used in distributed systems **to maintain a consistent order of events** across processes without requiring synchronized physical clocks

Logical Clock : used to maintain event ordering