🏠_Home_

Q

1. What Is Socket?
2. What Is Multiprogramming?
3. Explain the Advantages of Java Sockets & Disadvantages of Java Sockets.

4. How does TCP work?
5. Explain the difference between TCP and UDP protocol?

6. Brief on Client vs Server.
7. Explain the differences between Client and Server.
8. Describe client-server architecture using an appropriate diagram.

9. How do you write a multithreaded server in Java?
10. What is an ephemeral port?
11. Explain the following:
    a. IP Address
    b. Protocol
    c. Port Number
    d. MAC Address
    e. TCP
    f. FTP
    g. Telnet
    h. SMTP
    i. POP

12. Explain about Skeleton & Stub.
13. How do we Establish a Socket Connection?
14. What are the Important methods of socket & Server classes?
15. Why do we close created connections before the system Shutting Down?

A

1. **Socket**
   Network එකක run වෙන programme 2ක් අතර තියෙන two-way communication link එකේ, එක end point එකක්.

2. **Multiprogramming**
   system එකක multiple programs or processes හරිවිදියට run කරන්න තියෙන ක්‍රමයක්.
   (multithreading / multiprocessing ක්‍රම වලට implement කරන්න පුලුවන්)

3. **Advantages of Java Sockets**
   - Platform Independence
   - Simple and Flexible API
   - Wide Protocol Support
   - Robustness and Reliability
   - Scalability

   Disadvantages of Java Sockets
   - Low-Level Programming
   - Lack of Built-in Security
   - Limited Functionality
   - Performance Overhead
   - Asynchronous Operations

   (ChatGPT Full answer;

   Advantages of Java Sockets:

   1. **Platform Independence:** Java sockets provide a platform-independent networking solution, allowing communication between different devices and operating systems.

   2. **Simple and Flexible API:** Java sockets provide a straightforward and easy-to-use API, making it relatively

simple to implement network communication in Java applications.

3. **Wide Protocol Support:** Java sockets support various network protocols such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol), enabling developers to choose the appropriate protocol for their specific requirements.

4. **Robustness and Reliability:** Java sockets offer reliable and robust network communication, ensuring data integrity and error handling through features like automatic retransmission and congestion control.

5. **Scalability:** Java sockets support concurrent connections and can handle a large number of simultaneous client connections, making them suitable for developing scalable network applications.

Disadvantages of Java Sockets:

1. **Low-Level Programming:** Java sockets operate at a low level, requiring developers to handle low-level network details such as socket creation, binding, and managing input/output streams. This can make socket programming more complex and error-prone.

2. **Lack of Built-in Security**: Java sockets do not provide built-in security features, such as encryption and authentication. Developers need to implement additional security measures to ensure secure communication over sockets.

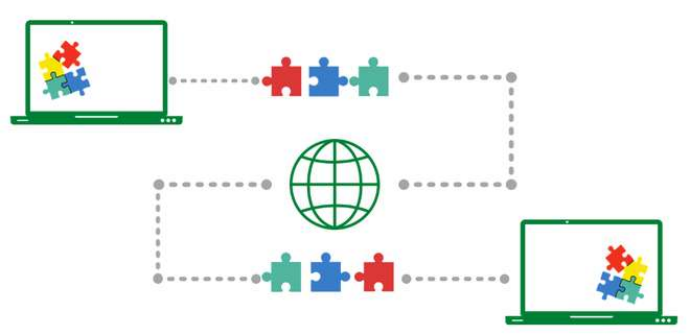
3. **Limited Functionality:** Java sockets primarily provide basic network communication capabilities. Additional functionality like handling higher-level protocols (HTTP, FTP) or advanced features (multicasting, broadcasting) may require more complex implementation or the use of additional libraries.

4. **Performance Overhead:** Java sockets introduce some performance overhead compared to lower-level networking libraries or languages due to the abstraction layers and additional features provided by the Java networking API.

5. **Asynchronous Operations:** Java sockets primarily support blocking I/O operations, which can limit the scalability and responsiveness of network applications. Implementing asynchronous socket operations requires additional programming techniques or using libraries like Java NIO (New I/O) or frameworks like Netty.

It's important to note that while Java sockets have certain disadvantages, they are still widely used and provide a solid foundation for network programming in Java. Additionally, there are alternative frameworks and libraries available in Java that can address some of the limitations mentioned above, providing higher-level abstractions and advanced features for network communication.)

4. TCP (Transmission Control Protocol)



Network එකක් හරහා විවිධ devices අතර පණිවිඩ හුවමාරු කර ගැනීමට උපකාර වන connection-oriented protocol එකක්.
IP network හරහා reliable and ordered (විශ්වාසදායක සහ පෙලගැස්වූ) දත්ත සම්ප්‍රේෂණය සිදුකරයි.

- client and a server අතර හොඳ connection එකක් ඇතිකිරීම
- Data කුඩා පැකට් වලට බෙදීම
- ordered(පෙලගැස්වූ) හා error-checked delivery
- Flow control එක පවත්වාගැනීම

- තදබදය පාලනය කිරීම
- Connection එක නිවැරදිව අවසන් කිරීම සිදුකරයි.

🔗What is Transmission Control Protocol (TCP)? - GeeksforGeeks

5. Differences between TCP and UDP

TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
connection-oriented protocol	connectionless protocol
• Reliable (විශ්වාසදායී) • Ordered (පෙලගැස්වූ) • Error-checked data transmission(සම්ප්‍රේශනය)	• unreliable • unordered • non-error-checked data transmission
Slow	faster
guarantees delivery	does not guarantee delivery
Flow control එක පවත්වාගනී.	එහෙම එකක් නෑ
තදබදය පාලනය කරයි.	නෑ.

🔗Differences between TCP and UDP - GeeksforGeeks

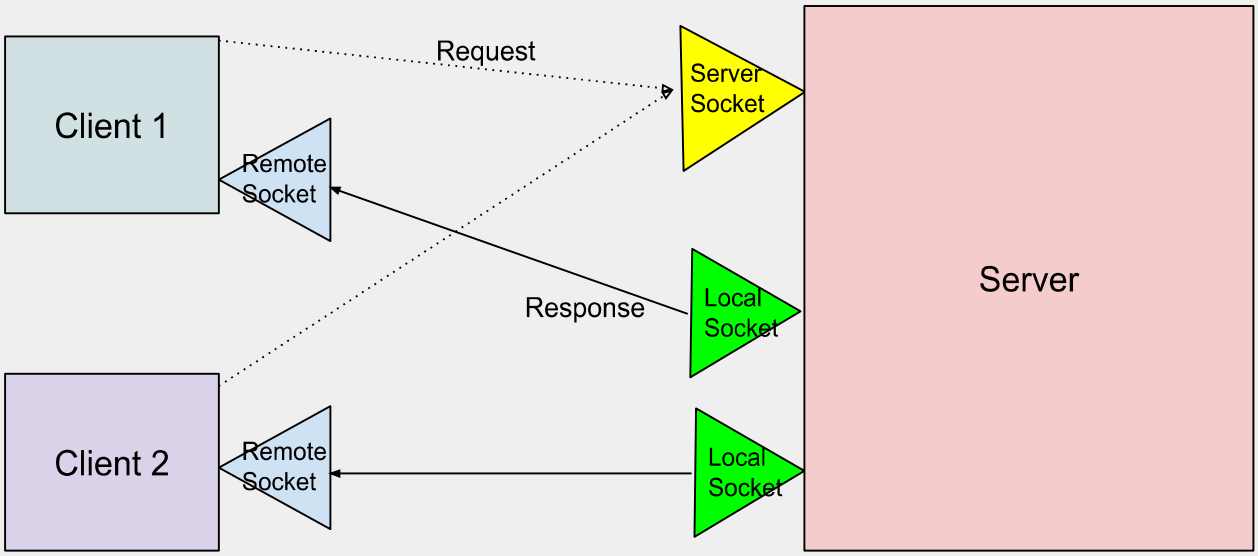
6. Brief on Client vs Server

Client	Server
Service එකක් / තොරතුරක් request කරයි.	Service එකක් / තොරතුරක් ලබා දේ.

7. Look 6 ⬆

8. client-server architecture

1. Client , Server එකේ **server socket** එකට **request** එකක් යවනො, service එකක් ඉල්ලලා.
2. Server එක ඇතුලෙ process එකක් වෙලා, client ට service එක සපයන්න local socket එකක් create කරනො server එකෙන්. (එක client කෙනෙක්ට, එක local socket එක ගානෙ)
3. **Local socket** එකෙන් client ට response එක යවනො.
4. ඒ response එක clientගෙ **remote socket** එකෙන් receive කරගන්නො.



9. To write a multithreaded server in Java:

1. Create a ServerSocket to listen for incoming client connections.
2. Accept client connections using the accept() method.
3. For each client connection, spawn a new thread.
4. Implement the thread's run() method to handle communication with the client.
5. Start the server by calling the start() method of the thread.
6. Implement a termination mechanism to gracefully shut down the server.

10. ephemeral/dynamic port (තාවකාලික port)

Client application එකට පිට network එක්ක communicate කරන්න , OS එකෙන් dynamically assign කරගන්න temporary port එකක්.

Session, connections, sockets, & ephemeral ports

11. Terminology

1. IP Address (Internet Protocol Address)

Computer network එකකට connect වුණු devices uniquely identify කරගන්න පුලුවන් address එකක්. (Software based,global identification)

2. MAC Address (Media Access Control address)

Computer network එකකට connect වුණු devices uniquely identify කරගන්න පුලුවන් address එකක්. (hardware based,local identification)

3. Protocol

Internet එක හරහා, data යවන සහ ලැබෙන ආකාරය control කරන Rules set එකක්.

4. Port Number

Computer එකක run වෙන specific application එකක් හෝ service එකක් identify කරගන්න number එක.

5. TCP (Transmission Control Protocol)

Reliable(විශ්වාසදායී), connection-oriented transmission(සම්ප්‍රේශන)protocol එකක්.

6. FTP (File Transfer Protocol)

Protocol for file transfer between client and server.

7. Telnet (Teletype network)

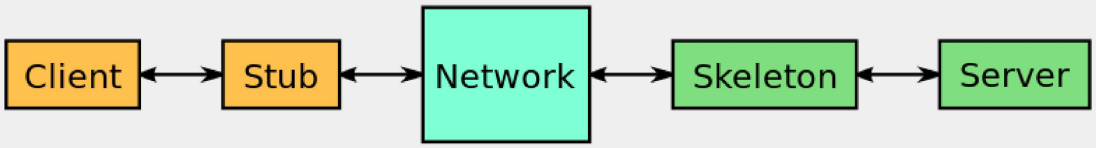
Protocol for remote command-line access to devices.

8. SMTP (Simple Mail Transfer Protocol)

Protocol for sending email between mail servers.

9. POP (Post Office Protocol)
Protocol for retrieving email from a mail server.

12. Skeleton & Stub



Stub: A client-side component that acts as a proxy for a remote object or service, providing a local interface for the client to invoke remote methods.
It handles the serialization of parameters, sends them to the server, and returns the result to the client.

Skeleton: A server-side component that receives incoming requests from clients, unpacks the requests, and dispatches them to the appropriate methods on the server.
It handles the deserialization of parameters, invokes the methods, and returns the results to the client through the stub.
It abstracts the low-level network communication details from the server application.

🔗 Java RMI - Introduction.
stub and skeleton in rmi | distributed system | Lec-24 | Bhan…

13. To establish a socket connection:

1. Create a Socket object, specifying the server's IP address or hostname and the port number.
2. Call the `connect()` method on the Socket object to connect to the server.
3. Obtain input and output streams from the Socket object to send and receive data.
4. Perform data exchange operations using the streams.
5. Close the socket connection by calling the `close()` method on the Socket object when finished.

14. Important methods of socket & Server classes

Socket Class:

1. `**Socket(String host, int port)**`: Creates a new socket and connects it to the specified host and port.
2. `**getInputStream()**`: Returns an InputStream object to read data from the socket.
3. `**getOutputStream()**`: Returns an OutputStream object to write data to the socket.
4. `**connect(SocketAddress endpoint)**`: Connects the socket to the specified endpoint.
5. `**close()**`: Closes the socket connection.
6. `**isConnected()**`: Returns true if the socket is currently connected.

ServerSocket Class:

1. `**ServerSocket(int port)**`: Creates a new ServerSocket that listens on the specified port.
2. `**accept()**`: Listens for incoming client connections and accepts them, returning a Socket object representing the client connection.
3. `**close()**`: Closes the server socket.
4. `**isBound()**`: Returns true if the server socket is currently bound to a port.
5. `**isClosed()**`: Returns true if the server socket is closed.

15. Closing connections before system shutdown is important to:

- Socket free කරල Network එකෙ performance වැඩි කරන්න.
- Release system resources associated with the connections.
- Allow for a graceful termination of network communication.
- Ensure efficient resource utilization.
- Maintain a consistent application state.

https://chat.openai.com/share/8d044389-5fd5-42ff-9d47-19c10bf15ee3

==========

Network ?

Data or Resources share කරගැනීමට, එකිනෙක connect කළ Computer 2ක් හෝ ඊට වැඩි ගණනක්.

Network Programming ?

Network එකක තියෙන devices අතර execute වන programs ලිවීම.

data stream?

Sequence of data.

Streams in socket programming

- Input stream - data read කරන්න
- Output stream - data write කරන්න

Thread

lightweight sub process / separate execution path

Full : MultiThread