Unit-1

1. How does client server architecture works?

   = Client-server architecture is a cornerstone of modern system design, where the network infrastructure is structured to include multiple clients and a central server. In this model, clients are devices or programs that make requests for services or resources, while the server is a powerful machine or software that fulfills these requests. Communication between clients and the server follows a request-response protocol, such as HTTP/HTTPS for web services or SQL for database queries.

   Here How It Works:

   - **Client Requests:** The client is usually a computer, phone, or any device that wants to access data or services. It sends a request to the server, like opening a website or logging into an app.
   - **Server Responds:** The server is a powerful computer that processes these requests. It checks the request, finds the needed data, and sends it back to the client.
   - **Communication:** The client and server communicate over a network (like the internet or a local network). This is done using protocols like **HTTP** (for websites) or **FTP** (for file transfer).

2. What are the factors to be considered when designing a software?

   = When designing software, several important factors must be considered to ensure it is efficient, scalable, and user-friendly.

   Here are the factors and they are listed below:

   - **Requirements Analysis** – Understanding user needs and software objectives.
   - **Scalability** – Ensuring the software can handle future growth.
   - **Security** – Protecting data from unauthorized access and threats.
   - **Performance** – Optimizing speed, responsiveness, and efficiency.

- **User Interface (UI) & User Experience (UX)** – Making the software easy and enjoyable to use.
- **Maintainability** – Writing clean, modular code for easy updates.
- **Compatibility** – Ensuring the software works across different devices and platforms.
- **Reliability** – Making sure the software functions correctly under different conditions.
- **Cost & Budget** – Planning resources, development, and maintenance costs.
- **Testing & Debugging** – Identifying and fixing errors before release.

3. Why do we really need network programming tools and platform? Explain some of them.

= Network programming tools and platforms are essential for developing, testing, and managing applications that communicate over networks. Network programming tools and platforms are crucial for developing secure, efficient, and scalable network applications. Whether for debugging, performance testing, or security analysis, they help developers and administrators maintain smooth network communication.

- **Efficient Communication** – Enable devices to send and receive data smoothly over networks like the internet.
- **Security Management** – Protect against cyber threats by encrypting and monitoring network traffic.
- **Performance Optimization** – Ensure faster data transfer and lower latency.
- **Debugging & Troubleshooting** – Help identify and fix network-related issues.
- **Scalability & Reliability** – Support the growth of applications while maintaining performance.

Here Are Some Important Network Programming Tools & Platforms Are Listed below:

- **Efficient Network Communication:** Tools like socket libraries provide a standardized way to create and manage network connections, ensuring efficient and reliable data exchange between applications.
- **Simplified Network Development:** Platforms like Node.js offer frameworks and libraries that streamline network application development, reducing complexity and development time.

- **Enhanced Security:** Tools like Wireshark help analyze network traffic, identify vulnerabilities, and implement security measures to protect sensitive data during transmission.
- **Troubleshooting and Debugging:** Network programming tools aid in identifying and resolving network issues, ensuring smooth and uninterrupted communication.
- **Performance Optimization:** Tools and platforms provide insights into network performance, allowing developers to optimize their applications for speed and efficiency.

Unit-3

1. What is URL? Give an example of a URL that shows each components of URL. Diagrammatic representation is more preferred.

   = A URL (Uniform Resource Locator) is a web address that provides a unique, specific location for a particular resource on the internet. It contains information about what you're looking for as well as the protocol used to access it. URLs are typically used to locate web pages, but they can also be used to locate other resources such as images, videos, audio files and documents. In other words, URLs make it possible to identify where something is located online so that you can view or download it.

   Example URL:

   https://www.example.com/resources/articles/page.html?search=topic&page=2#section3

   Breakdown of Components:

   - **Protocol (https):** Specifies the protocol used to access the resource (e.g., HTTP, HTTPS, FTP). HTTPS indicates a secure connection.
   - **Domain Name (www.example.com):** The human-readable name of the website's server.
   - **Path (/resources/articles/page.html):** Specifies the location of the resource on the server.
   - **Query Parameters (?search=topic&page=2):** Optional parameters that provide additional information to the server (e.g., search terms, page number).
   - **Fragment Identifier (#section3):** Optional identifier that points to a specific section or element within the resource.

2. URLs vs URIs with examples for each. Can all URLs be URIs? Why?

   = The difference between of URLs vs URIs with examples for each are:

URL	URI
URL is used to describe the identity of an item.	URI provides a technique for defining the identity of an item.
URL links a web page, a component of a web page or a program on a web page with the help of accessing methods like protocols.	URI is used to distinguish one resource from other regardless of the method used.
URL provides the details about what type of protocol is to be used.	URI doesn't contains the protocol specification.
URL is a type of URI.	URI is the superset of URL.
It comprises of protocol, domain, path, hash, and so on.	It comprises of scheme, authority, path, query and many more.
Ex-https://www.geeksforgeeks.org/	Ex- urn:isbn:0-294-56559-3

Yes, all URLs are URIs. This is because a URL is a specific *type* of URI. It's a subset of the broader URI concept. A URL provides the location and access method of a resource, which also serves to identify it.