UNIT-1 INTRODUCTION

1.1 Network Programming Features and Scope

1.2 Network Programming Language, Tools and Platforms

1.3 Client and Server Application

1.4 Client server model and software design

NETWORK PROGRAMMING FEATURES AND SCOPE

- Network programming is the process of creating software that is capable of communicating with other devices over a network. This involves the use of various networking protocols, such as TCP/IP, UDP, and HTTP, as well as programming interfaces and libraries that provide access to the underlying network stack.
- Network programming can be used to develop a wide range of applications, including client-server systems, peer-to-peer systems, and distributed systems. These applications may include web servers, email clients, instant messaging systems, and file-sharing programs, among others. Network programming also encompasses the areas of network security, network management, and network optimization.
- Network programming is typically done using sockets, which are a standard way for networked devices to communicate with each other. Sockets can be used to transmit data using different protocols, such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Network programming also involves working with different layers of the OSI (Open Systems Interconnection) model, which defines how data is transmitted over a network.

NETWORK PROGRAMMING FEATURES

- **Communication**: Network programming allows devices to communicate with each other by sending and receiving data over a network.
- **Concurrency**: Network programming often involves multiple devices or threads communicating simultaneously, which requires the use of concurrency techniques such as threading or asynchronous programming.
- **Network protocols**: Network programming involves working with different network protocols, such as TCP, UDP, HTTP, FTP, and SSH, to establish and maintain network connections.
- **Sockets**: Sockets are a common feature of network programming, as they provide a standard way for devices to communicate with each other over a network.

  A socket is a endpoint for sending or receiving data across a computer network. It is a combination of an IP address and a port number that is used to identify a specific process running on a device. Sockets provide a common interface for network communication in many operating systems, allowing different programs and devices to communicate with each other using a standard set of rules.
- **Security**: Network programming often includes the implementation of security measures, such as encryption and authentication, to protect against network-based attacks.

NETWORK PROGRAMMING FEATURES

- **Network Management**: Network programming includes managing the network connections and configurations, such as IP addresses, ports, and routing tables.
- **Error Handling:** Network programming requires to handle errors that may occur during communication, such as lost connections or invalid packets, and to implement appropriate recovery mechanisms.
- **Scalability**: Network programming often involves designing systems that can handle a large number of connections or handle high traffic loads.
- **Portability**: Network programming often requires that the program can run on multiple platforms or devices and can interact with other systems.
- **Optimization**: Network programming is often required to optimize the performance of the system, such as reducing latency, increasing throughput, and minimizing resource usage.

NETWORK PROGRAMMING SCOPE

- **Client-Server Systems**: Network programming is used to create client-server systems, where a client device sends requests to a server and the server responds with the appropriate information. Examples include web servers, email servers, and file servers.
- **Distributed Systems**: Network programming is used to create distributed systems, where multiple devices work together to perform a task. Examples include distributed computing systems, peer-to-peer networks, and grid computing.
- **Network protocols**: Network programming is used to implement, maintain and improve network protocols such as TCP, UDP, HTTP, FTP, SSH, DNS and many more.
- **Network Security**: Network programming is used to implement security measures, such as encryption and authentication, to protect against network-based attacks.
- **Internet of Things** (IoT): Network programming is used to create applications for IoT devices, such as home automation systems and industrial control systems, to communicate with other devices over a network.
- **Network Management**: Network programming is used to manage network connections and configurations, such as IP addresses, ports, and routing tables.
- **Cloud Computing**: Network programming is used to create applications for cloud computing, such as distributed storage systems and distributed computing systems, that use network communication to share resources and data.
- **Machine Learning and Artificial Intelligence**: Network programming is used to create distributed systems that leverage machine learning and artificial intelligence techniques to process data and make predictions.
- **Mobile and Wireless Networking**: Network programming is used to create mobile and wireless networking applications, such as mobile apps, wireless sensor networks, and cellular networks.

NETWORK PROGRAMMING SCOPE

- **Gaming**: Network programming is used to create online multiplayer games, where players can interact with each other in real-time over a network.
- **Streaming**: Network programming is used to create streaming services, such as video and audio streaming, where a server streams media to multiple clients over a network.
- **Virtual Private Network** (VPN): Network programming is used to create VPNs, which allow users to securely access a private network over a public network, such as the internet.
- **Remote Access**: Network programming is used to create remote access solutions, such as remote desktop and virtual private network (VPN) connections, that allow users to access resources on a remote network.
- **Network Monitoring**: Network programming is used to create tools for monitoring and analyzing network traffic, such as packet sniffers and network analyzers.
- **Network Automation**: Network programming is used to create tools for automating network management tasks, such as configuring network devices and monitoring network performance.
- **Network Simulation**: Network programming is used to create tools for simulating network behavior, such as traffic generators and network emulators, which are used for testing and research purposes.

NETWORK PROGRAMMING LANGUAGE

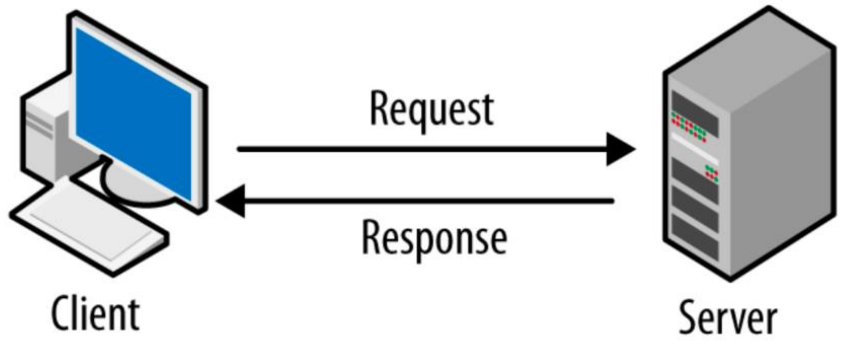
- Network programming languages are programming languages that are specifically designed to create software that runs on, interacts with, and manages networked systems. These languages typically provide libraries and API's for working with network protocols, sockets, and other networking concepts. These languages are used to create a wide range of network-based applications such as servers, clients, network management tools, network security software, and more.
- C: A low-level programming language that is widely used for network programming due to its efficiency and ability to work directly with sockets.
- Python: A high-level programming language that is often used for network programming because of its simple syntax and built-in libraries for working with sockets.
- Java: A cross-platform programming language that is often used for network programming because of its built-in support for sockets and its ability to run on many different platforms.
- C#: A modern, high-level programming language that is often used for network programming because of its built-in support for sockets and its ability to run on the Microsoft .NET Framework.
- JavaScript: A popular programming language that is used for creating web applications, it is often used in conjunction with other technologies such as node.js, WebSocket to create real-time applications such as chat apps, and games.
- Perl, Ruby, and PHP are also used for network programming, but not as common as the above-mentioned languages.
- In summary, network programming languages are a set of programming languages that provide a set of libraries and APIs for creating applications that interact with the network. These languages are used to create a wide range of network-based applications such as servers, clients, network management tools, network security software, and more

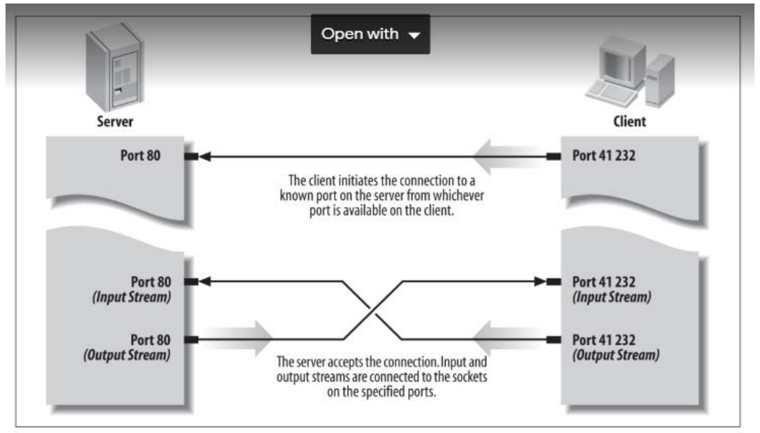
TOOLS AND PLATFORMS USED FOR NETWORK PROGRAMMING:

- **Sockets**: Sockets are a common API for network programming that allows applications to send and receive data over a network using a variety of protocols. Sockets are available in many programming languages including C, C++, Python, Java, and C#.
- **Network protocols libraries**: Network protocols libraries are libraries that provide an API for working with specific network protocols. Examples include libraries for working with TCP, UDP, HTTP, FTP, SSH, DNS and many more.
- **Network simulation tools**: Network simulation tools are used to simulate network behavior in order to test and research network-based systems. Examples include NS-3, Mininet, and GNS3.
- **Network monitoring and analysis tools**: Network monitoring and analysis tools are used to monitor and analyze network traffic in order to troubleshoot and optimize network performance. Examples include Wireshark, tcpdump, and Nagios.
- **Network management tools**: Network management tools are used to manage network connections and configurations. Examples include ifconfig, route, and iptables.
- **Cloud platforms**: Cloud platforms such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) provide a range of network-related services, including virtual private networks (VPNs), load balancers, and content delivery networks (CDNs).
- **Virtualization platforms**: Virtualization platforms such as VMware and VirtualBox allow network administrators to create virtual networks and test network-based systems in a controlled environment.
- **Network Automation tools**: Network Automation tools such as Ansible, Puppet, and Chef are used to automate network management tasks such as configuring network devices and monitoring network performance.
- **IoT platforms**: IoT platforms such as AWS IoT Core and Microsoft Azure IoT provide a range of services for developing and deploying IoT applications, including device management, data storage, and message queues.

A CLIENT-SERVER APPLICATION

- A client-server application is a type of software system in which one or more clients connect to a central server in order to access shared resources or services. The client is typically a small, lightweight program that runs on a user's computer or mobile device, while the server is a more powerful computer that manages the shared resources and handles client requests. Examples of client-server applications include email systems, web browsers, and online gaming platforms.
- A client-server application is a type of software system that consists of two main components: a client and a server. The client is a program that runs on a user's device and is used to connect to and access the resources or services provided by the server. The server is a program that runs on a separate device and is responsible for managing the shared resources and handling requests from the clients.
- The client and server communicate with each other through a network, such as the Internet or a local area network (LAN). Examples of client-server applications include email systems, web browsers, and online gaming platforms.





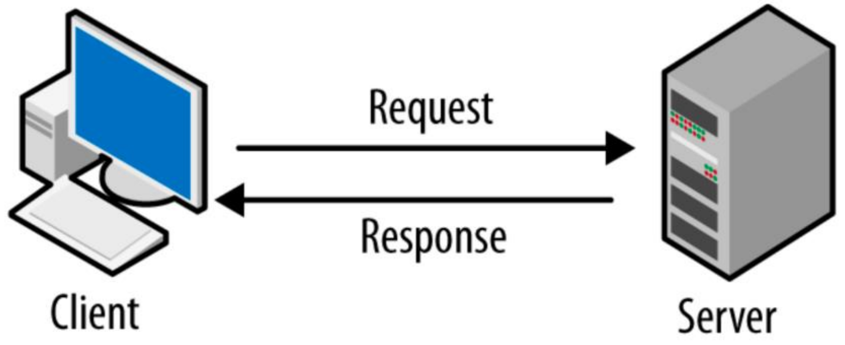
The client initiates the connection to a known port on the server from whichever port is available on the client.

The server accepts the connection. Input and output streams are connected to the sockets on the specified ports.

Figure 1-5. A client/server connection

A CLIENT-SERVER APPLICATION

- A client-server application is a type of software system in which one or more clients connect to a central server in order to access shared resources or services. The client is typically a small, lightweight program that runs on a user's computer or mobile device, while the server is a more powerful computer that manages the shared resources and handles client requests. Examples of client-server applications include email systems, web browsers, and online gaming platforms.



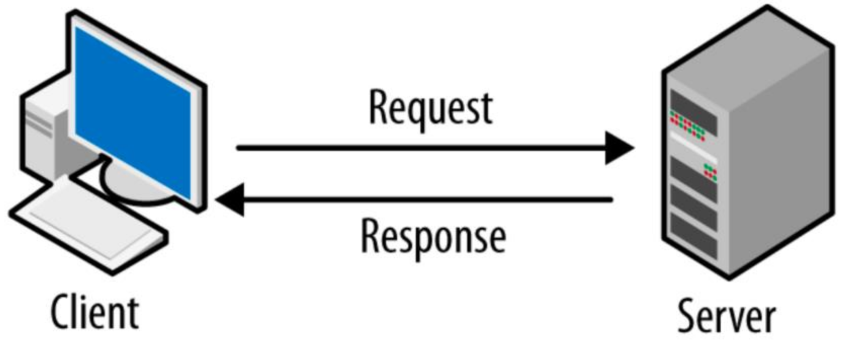
Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

A CLIENT-SERVER APPLICATION

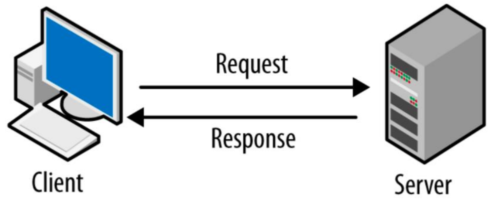


➢ How the browser interacts with the servers ?

1. User enters the URL(Uniform Resource Locator) of the website or file. The Browser then requests the DNS(www.fb.com) Server.
2. DNS Server lookup for the address of the WEB Server.
3. DNS Server responds with the IP address of the WEB Server.
4. Browser sends over an HTTP/HTTPS request to WEB Server's IP (provided by DNS server).
5. Server sends over the necessary files of the website.
6. Browser then renders the files and the website is displayed.

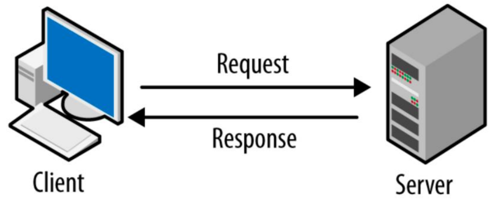
This rendering is done with the help of DOM (Document Object Model) interpreter, CSS interpreter and JS Engine collectively known as the JIT or (Just in Time) Compilers.

A CLIENT-SERVER MODEL AND SOFTWARE DESIGN



- The client-server model is a software design pattern that is based on the distributed computing architecture of the same name. It involves separating the client and server components of an application and having them communicate with each other over a network. This separation allows for a clear division of responsibilities between the client and server, and allows for greater flexibility, scalability, and ease of maintenance in the overall software design.
- In terms of software design, the client component is responsible for handling the user interface and user interactions, while the server component is responsible for managing shared resources and handling requests from the clients. The client component typically runs on a user's device, such as a personal computer or mobile device, while the server component typically runs on a separate device or a network of devices.
- One of the key advantages of the client-server model is that it allows for a more efficient use of resources. The server can handle multiple requests simultaneously and handle computation-intensive tasks, while the client can handle the user interface and user interactions. Additionally, the client-server model allows for easier security management as the server can handle the authorization and authentication of clients.
- Overall, the client-server model is a widely used and powerful software design pattern that is well suited for many types of software systems that require a central point of control and management for shared resources. It is also the foundation for many modern technologies such as cloud computing, mobile apps, and the internet of things (IoT), which rely on the ability to connect multiple devices and clients to a central server in order to access and share resources.

A CLIENT-SERVER MODEL AND SOFTWARE DESIGN



- Another advantage of the **client-server model** is that it allows for a more modular and scalable design. The server can be upgraded or replaced independently of the clients, and new clients can be added without affecting the functioning of existing clients or the server. This makes it easier to update and maintain the software over time.
- In terms of implementation, the client-server model typically uses a request-response protocol, where the client sends a request to the server, and the server responds with the requested resource or service. This communication is typically done using standard protocols such as HTTP, HTTPS, and TCP/IP.
- In conclusion, the client-server model is a powerful software design pattern that is widely used in many types of software systems. It allows for a clear separation of concerns between the client and server components, and allows for greater flexibility, scalability, and ease of maintenance in the overall software design.