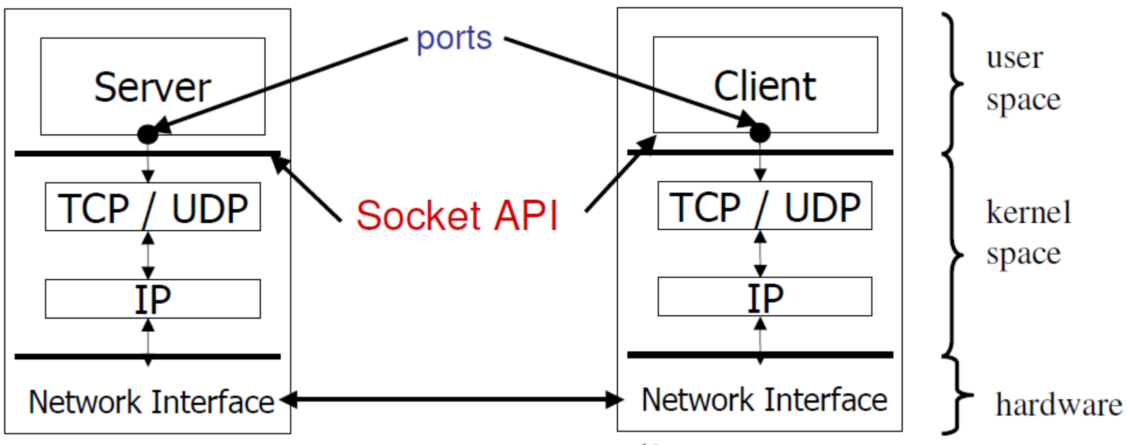
Assignment on Socket Programming in C for TCP and UDP

Aim The goal of this assignment is to explore socket programming in C for both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). The assignment will detail how to use socket functions, understand socket structures, and implement both types of socket programming for communication between client and server applications.

Socket programming enables communication between two machines over a network. It allows processes running on different systems to exchange data in a standardized way. TCP and UDP are two of the most common protocols for network communication.

- **TCP** is connection-oriented, ensuring reliable data transfer with flow control, error detection, and retransmission of lost data packets.
- **UDP** is connectionless, providing faster data transfer but without guarantees for delivery or error correction.

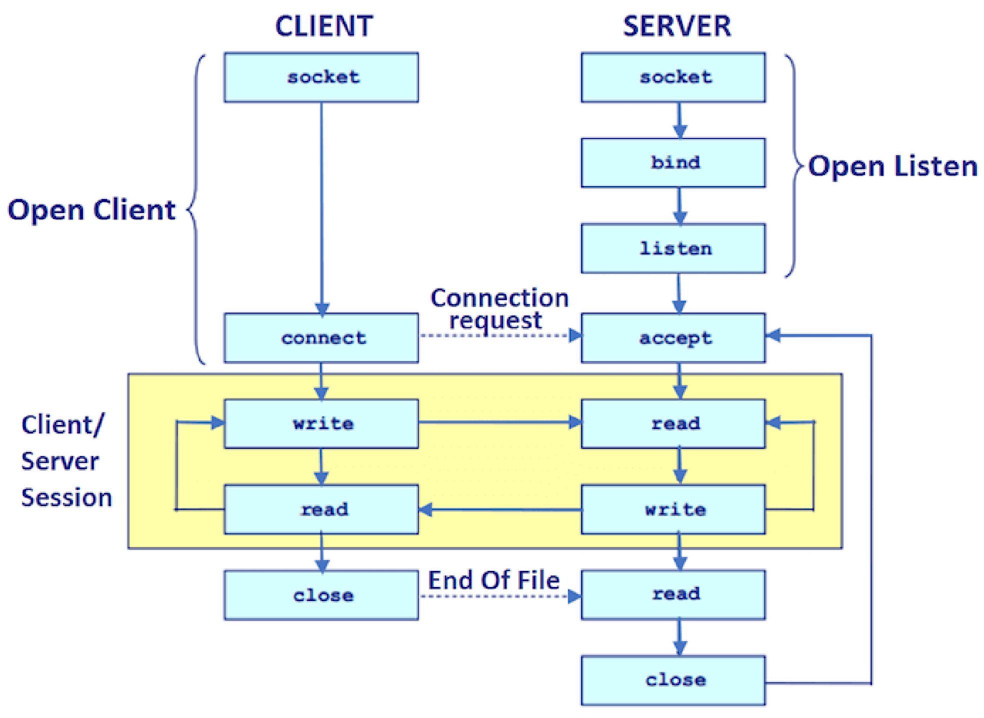
A **socket** is an endpoint for sending or receiving data across a computer network. In C programming, we use various socket functions to establish communication. A socket has an associated **address structure** that contains information such as the protocol (TCP/UDP), IP address, and port number.



* Port numbers: 1,024 -- 65,535 (2^{16} - 1)

Key Socket Functions:

1. socket() – Creates a socket.
2. bind() – Binds a socket to a specific address (IP address and port).
3. listen() – Prepares a server socket to accept connections (for TCP).
4. accept() – Accepts an incoming connection (for TCP).
5. connect() – Initiates a connection to a remote socket (for TCP).
6. send() / recv() – Sends and receives data over a socket (for both TCP and UDP).
7. sendto() / recvfrom() – Used for sending and receiving data on UDP sockets.
8. close() – Closes the socket when done.



SOCKET API

Socket Types

1. **TCP Socket (Stream Socket)**:
    - Used for reliable communication.
    - Connection-oriented.
    - Supports flow control, error recovery, and retransmission.
2. **UDP Socket (Datagram Socket)**:
    - Used for fast, but unreliable communication.
    - Connectionless.
    - No error handling or retransmission.

4. Socket Structures in C

A socket is represented by an instance of the sockaddr structure, which is a generic structure used for both TCP and UDP. However, there are different specific address structures based on the protocol (IPv4, IPv6).

sockaddr_in (IPv4)

For TCP and UDP communication, the most common address structure is sockaddr_in. This structure is used when working with IPv4 addresses.

```
struct sockaddr_in {
  sa_family_t    sin_family;  // Address family (AF_INET for IPv4)
  in_port_t      sin_port;    // Port number (in network byte order)
  struct in_addr sin_addr;    // IPv4 address
};
```

- **sin_family**: Specifies the address family. For IPv4, it is AF_INET.
- **sin_port**: Port number in network byte order.
- **sin_addr**: IPv4 address in the form of in_addr structure.

in_addr is defined as:

```
struct in_addr {
    uint32_t s_addr;  // IPv4 address (32 bits)
};
```

sockaddr_in6 (IPv6)

For IPv6, the structure is sockaddr_in6:

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;   // Address family (AF_INET6 for IPv6)
    in_port_t       sin6_port;     // Port number (in network byte order)
    uint32_t        sin6_flowinfo; // Flow information (for future use)
    struct in6_addr sin6_addr;     // IPv6 address
    uint32_t        sin6_scope_id; // Scope ID for link-local addresses
};
```

- **sin6_family**: Address family (AF_INET6 for IPv6).
- **sin6_port**: Port number in network byte order.
- **sin6_addr**: IPv6 address in in6_addr structure.

TCP Socket Programming

The following steps outline the process of TCP socket programming:

1. Creating a Socket

A socket is created using the socket() function. The first argument specifies the address family (AF_INET for IPv4), the second specifies the socket type (SOCK_STREAM for TCP), and the third argument specifies the protocol (0 for default).

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

2. Binding the Socket

The server binds the socket to a specific IP address and port using the bind() function. This function takes three arguments: the socket file descriptor, the address structure, and the length of the address.

```
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(8080);   // Convert port to network byte order
server_addr.sin_addr.s_addr = INADDR_ANY; // Bind to any available IP address

bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
```

3. Listening for Connections

The server listens for incoming connections using the listen() function. The second argument specifies the number of connections that can be queued before the server accepts the connection.

```
listen(sockfd, 5);
```

4. Accepting Connections

The server accepts incoming connections using the accept() function, which returns a new socket descriptor for the accepted connection.

```
int new_sock = accept(sockfd, (struct sockaddr *)&client_addr, &client_addr_len);
```

5. Data Transfer

Once the connection is established, data can be sent or received using send() and recv() functions.

```
send(new_sock, message, strlen(message), 0);

recv(new_sock, buffer, sizeof(buffer), 0);
```

6. Closing the Socket

After communication is finished, close the socket using the close() function.

```
close(sockfd);

close(new_sock);
```

---

6. UDP Socket Programming

UDP socket programming follows a different approach because it is connectionless. The main difference is that the server does not listen or accept connections. Instead, it simply receives data from clients.

1. Creating a Socket

A UDP socket is created in the same way as a TCP socket, but the socket type is SOCK_DGRAM.

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

2. Binding the Socket

The server binds the socket to a port, similar to TCP. The difference is that we do not listen for connections.

```
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
```

3. Sending Data

Data is sent using sendto() function. It takes the socket, the message, the size of the message, the destination address, and the size of the destination address.

```
sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&client_addr,
sizeof(client_addr));
```

4. Receiving Data

Data is received using recvfrom(), which provides the source address of the message.

```
recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&client_addr,
&client_addr_len);
```

5. Closing the Socket

Finally, close the socket when communication is finished.

```
close(sockfd);
```

Comparison:

Feature	**TCP**	**UDP**
Connection	Connection-oriented	Connectionless
Reliability	Reliable, ensures delivery	Unreliable, no guarantee
Speed	Slower due to overhead	Faster, less overhead
Use Cases	Web, email, file transfer	Streaming, gaming, VoIP
Error Checking	Yes, with recovery	Basic checksum, no recovery
Flow Control	Yes	No
Header Size	Larger (20 bytes)	Smaller (8 bytes)
Order of Delivery	Guarantees correct order	No guarantee of order

Conclusion

In this Assignment we covered the basic concepts of socket programming in C using both TCP and UDP protocols. TCP provides reliable communication, while UDP offers a faster, connectionless approach. By understanding socket functions and structures like sockaddr_in and sockaddr_in6, we can effectively implement network communication for different use cases.