R (2)	N (5)	Oral (3)	Total (2)	Dated Sign

Assignment Group- A_07

Problem Definition

Write a program using TCP socket for wired network for following

- Say Hello to Each other (For all students)
- File transfer (For all students)
- Calculator (Arithmetic) (50% students)
- Calculator (Trigonometry) (50% students)

Demonstrate the packets captured traces using Wire shark Packet Analyzer Tool for peer to peer mode..

1.1 Prerequisite:

- IP Address and OSI & TCP/IP Model.
- Role of different servers.
- Basic C programming.

1.2 Learning Objectives:

- Understand data exchange features of Sockets.
- Understand server client socket programming.

1.3 Theory

Introduction

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

Socket Structures

Socket address structures are an integral part of every network program. We allocate them, fill them in, and pass pointers to them to various socket functions. Sometimes we pass a pointer to one of these structures to a socket function and it fills in the contents.

We always pass these structures by reference (i.e., we pass a pointer to the structure, not the structure itself), and we always pass the size of the structure as another argument.

When a socket function fills in a structure, the length is also passed by reference, so that its value can be updated by the function. We call these value-result arguments.

Always, set the structure variables to NULL (i.e., '\0') by using memset() for bzero() functions, otherwise it may get unexpected junk values in your structure.

Where is Socket Used?

A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

2.1 Socket Types.

There are four types of sockets available to the users. The first two are most commonly used and the last two are rarely used.

Processes are presumed to communicate only between sockets of the same type but there is no restriction that prevents communication between sockets of different types.

- **Stream Sockets** − Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries. For declaring TCP socket **SOCK_STREAM** is used.
- **Datagram Sockets** − Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol). **SOCK_DGRAM** Supports datagram (connectionless, unreliable messages of a fixed maximum length).
- **Raw Sockets** − These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol. **SOCK_RAW** Provides raw network protocol access.
- **Sequenced Packet Sockets** −. **SOCK_SEQPACKET** Provides a sequenced, reliable, two-way connection based data transmission path for datagram of fixed maximum length; a consumer is required to read an entire packet with each input system call.

2.2 Addressing, Protocol Families and Socket Types

A socket is one endpoint of a communication channel used by programs to pass data back and forth locally or across the Internet. Sockets have two primary properties controlling the way they send data: the address family controls the OSI network layer protocol used and the socket type controls the transport layer protocol.

C supports three address families.

- **AF_INET** is used for IPv4 Internet addressing. IPv4 addresses are made up of four octal values separated by dots (e.g., 10.1.1.5 and 127.0.0.1). These values are more commonly referred to as “IP addresses.” Almost all Internet networking is done using IP version 4 at this time.

- **AF_INET6** is used for IPv6 Internet addressing. IPv6 is the "next generation" version of the Internet protocol, and supports 128-bit addresses, traffic shaping, and routing features not available under IPv4. Adoption of IPv6 is still limited, but continues to grow.
- **AF_UNIX** is the address family for Unix Domain Sockets (UDS), an inter-process communication protocol available on POSIX-compliant systems.

2.3 Protocol − The argument should be set to the specific protocol type given below, or 0 to select the system's default for the given combination of family and type −

Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

3.1 Network Byte Orders

Unfortunately, not all computers store the bytes that comprise a multisystem value in the same order. Consider a 16-bit internet that is made up of 2 bytes. There are two ways to store this value.

- **Little Endian** − In this scheme, low-order byte is stored on the starting address (A) and high-order byte is stored on the next address (A + 1).
- **Big Endian** − In this scheme, high-order byte is stored on the starting address (A) and low-order byte is stored on the next address (A + 1).

To allow machines with different byte order conventions communicate with each other, the Internet protocols specify a canonical byte order convention for data transmitted over the network. This is known as Network Byte Order.

While establishing an Internet socket connection, you must make sure that the data in the sin_port and sin_addr members of the sockaddr_in structure are represented in Network Byte Order

3.2 Byte Ordering Functions

Routines for converting data between a host's internal representation and Network Byte Order are as follows −

Function	Description
htons()	Host to Network Short
htonl()	Host to Network Long
ntohl()	Network to Host Long
ntohs()	Network to Host Short

Listed below are some more detail about these functions −

- **unsigned short htons(unsigned short hostshort)** − This function converts 16-bit (2-byte) quantities from host byte order to network byte order.

- **unsigned long htonl(unsigned long hostlong)** − This function converts 32-bit (4-byte) quantities from host byte order to network byte order.
- **unsigned short ntohs(unsigned short netshort)** − This function converts 16-bit (2-byte) quantities from network byte order to host byte order.
- **unsigned long ntohl(unsigned long netlong)** − This function converts 32-bit quantities from network byte order to host byte order.

3.3 IP Address Functions

UNIX provides various function calls to help you manipulate IP addresses. These functions convert Internet addresses between ASCII strings (what humans prefer to use) and network byte ordered binary values (values that are stored in socket address structures).

The following three function calls are used for IPv4 addressing –

- int inet_aton(const char *strptr, struct in_addr *addrptr)
- in_addr_t inet_addr(const char *strptr)
- char *inet_ntoa(struct in_addr inaddr)

4.1 Socket - Core Functions

4.2 The socket Function

To perform network I/O, the first thing a process must do is, call the socket function, specifying the type of communication protocol desired and protocol family, etc.

int socket (int family, int type, int protocol);

This call returns a socket descriptor that you can use in later system calls or -1 on error.

Parameters

Family − It specifies the protocol family and is one of the constants shown below –

Family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing Sockets
AF_KEY	Ket socket

Type− It specifies the kind of socket you want. It can take one of the following values –

Type	Description
SOCK_STREAM	Stream socket
SOCK_DGRAM	Datagram socket
SOCK_SEQPACKET	Sequenced packet socket
SOCK_RAW	Raw socket

Protocol − The argument should be set to the specific protocol type given below, or 0 to select the system's default for the given combination of family and type –

Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

4.1.1 .The connect Function

The connect function is used by a TCP client to establish a connection with a TCP server. If the socket has not already been bound to a local address, connect() shall bind it to an address which, unless the socket's address family is AF_UNIX, is an unused local address.

This call returns 0 if it successfully connects to the server, otherwise it returns -1 on error.

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

Parameters

1. **sockfd** − It is a socket descriptor returned by the socket function.
2. **serv_addr** − It is a pointer to struct sockaddr that contains destination IP address and port.
3. **addrlen** − Set it to sizeof(struct sockaddr).

4.1.2. The bind Function

The bind function assigns a local protocol address to a socket. With the Internet protocols, the protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number. This function is called by TCP server only.

int bind(int sockfd, struct sockaddr *my_addr,int addrlen);

This call returns 0 if it successfully binds to the address, otherwise it returns -1 on error.

Parameters

- **sockfd** − It is a socket descriptor returned by the socket function.
- **my_addr** − It is a pointer to struct sockaddr that contains the local IP address and port.
- **addrlen** − Set it to sizeof(struct sockaddr).

You can put your IP address and your port automatically

A 0 value for port number means that the system will choose a random port, and INADDR_ANY value for IP address means the server's IP address will be assigned automatically.

server.sin_port = 0;

server.sin_addr.s_addr = INADDR_ANY;

All ports below 1024 are reserved. You can set a port above 1024 and below 65535 unless they are the ones being used by other programs.

4.1.3 The listen Function

The listen function is called only by a TCP server and it performs two actions −

The listen function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.

The second argument to this function specifies the maximum number of connections the kernel should queue for this socket.

int listen(int sockfd,int backlog);

This call returns 0 on success, otherwise it returns -1 on error.

Parameters

- **sockfd** − It is a socket descriptor returned by the socket function.
- **backlog** − It is the number of allowed connections.

4.1.4 The accept Function

The accept function is called by a TCP server to return the next completed connection from the front of the completed connection queue. The signature of the call is as follows −

int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);

This call returns a non-negative descriptor on success, otherwise it returns -1 on error. The returned descriptor is assumed to be a client socket descriptor and all read-write operations will be done on this descriptor to communicate with the client.

Parameters

- **sockfd** − It is a socket descriptor returned by the socket function.
- **cliaddr** − It is a pointer to struct sockaddr that contains client IP address and port.
- **addrlen** − Set it to sizeof(struct sockaddr).

4.1.5 The send Function

The send function is used to send data over stream sockets or CONNECTED datagram sockets. If you want to send data over UNCONNECTED datagram sockets, you must use sendto() function.

You can use write() system call to send data. Its signature is as follows −

int send(int sockfd, const void *msg, int len, int flags);

This call returns the number of bytes sent out, otherwise it will return -1 on error.

Parameters

- **sockfd** − It is a socket descriptor returned by the socket function.
- **msg** − It is a pointer to the data you want to send.
- **len** − It is the length of the data you want to send (in bytes).
- **flags** − It is set to 0.

4.1.6 The recv Function

The recv function is used to receive data over stream sockets or CONNECTED datagram sockets. If you want to receive data over UNCONNECTED datagram sockets you must use recvfrom().

You can use read() system call to read the data. This call is explained in helper functions chapter.

int recv(int sockfd, void *buf, int len, unsigned int flags);

This call returns the number of bytes read into the buffer, otherwise it will return -1 on error.

Parameters

- **sockfd** − It is a socket descriptor returned by the socket function.
- **buf** − It is the buffer to read the information into.
- **len** − It is the maximum length of the buffer.
- **flags** − It is set to 0.

Assignment Question:

1. What is Socket and where it is used?
2. What is the deference between TCP and UDP SOCKET?
3. Why bit ordering is used?

Conclusion:

Hence we conclude that we have to create socket for transferring any data between the source and destination using socket programming we transfer TCP and UDP data. We also observed this communication using net stat command or wire shark packet analyser