DATE:06.08.24

SOCKET PROGRAMING
WEEK-1

<u>What is socket programming?</u>

Socket programming is a way of connecting two nodes on a networks to communicate with one another.

<u>Types of sockets?</u>

There are two types of sockets TCP(transfer control protocol) AND UDP(user datagram protocol).
TCP(Transmission control protocol):

- Connection based protocol which uses 3-way handshake between client and server .
- Here the *server* is kept running (through **".listen"** method),client sends a connection request to the server via **".connect"** method and server acknowledges the connection and data transfer takes place through this established connection stream.
- The first data packet contains the destination address.

UDP(user datagram protocol):

- Datagram oriented protocol with no acknowledgement of data transfer.
- Here each datapacket contains a the destination address. Each data packets follow a different path and reach the destination through a different path.

TCP(transfer control protocol)	**UDP(user datagram protocol)**
Connection-oriented	*Connectionless*
Reliable(ensures data is received in order)Order in which *data received is guaranteed.*	*Unreliable*(no guarantee of the data itself)No *guarantee of order* of data.
Dtata transmission is *stream based*(there is continuous flow of data)	Datagram-based(individual packets) Datagram-refers to self-conatined,*independent packets* of data that is sent over a network.
Connection setup happens in a *3WAY handshake manner*. (SYN,SYN-ACK,ACK)	No connection setup required.
Automatic retransmission of lost packets.	No retransmission.
Higher overhead due to reliability,hence *slow*	*Lower overhead*,hence *fast*
Ex:HTTPS,HTTP,FTP **USE:**suitable for application where reliablity and data integrity is critical. Ie)web browsing,email,file transfer.	**Ex:*DNS-Domain name system*** **USE:** suitable when speed is critical and occasional data loss is acceptable.ie)gaming,video streaming.

TCP-SERVER SIDE:

1. Create a TCP socket.*socket ( )*
2. Bind the socket to a specific IP address and port.*bind ()*
3. Listen for incoming connections.*listen( )*
4. Accept a connection from a client.*accept(*){Accept till connection made to client by server}
5. Datat tranfer:{echo program:Read data from the client and send the same data back (echo it).}*send( )* and *receive( )*
6. Close the connection. *Close( )*

TCP-CLIENT SIDE:

1. Create a TCP socket.*socket( )*
2. Connect to the server using its IP address and port.*connect( )*
3. Send data to the server.*send( )*
4. Receive the echoed data from the server.*recieve( )*
5. Close the connection.*close( )*

FUNCTIONS:

1.Socket( )

- It's present in <sys/socket.h> header file
- It is used to create a socket.
- **SYNTAX:** *int* socket(*int* family, *int* type, *int* protocol)
- Domain/addres Family-protovol family used for communication.
  - *AF_INET* : IPv4
  - *AF_INET6* :IPv6
  - *AF_UNIX/AF_LOCAL*:for local communication
- Type-it tells whether it will be TCP/UDP
  - *SOCK_STREAM*: TCP, connection oriented byte stream.
  - *SOCK_DGRAM*: UDP, connectionless message.
- Protocol-it tells what protocol(rules) to be used with the socket.Set to zero!
- Return value-it's a file descriptor(an integer),return -1 if it fails and 'errno' is set to indicate the error.**File descriptor** is an integer that uniquely identifies an open file or socket within a process.

2.connect( )

- It helps to establish connection between a client and a server.
- It's "used **on** the client side" to initiate connection "**to** a server".
- **SYNTAX:** *int connect (int sockfd, const struct sockaddr *addr, socklen_t addrlen);*
- *Sockfd:*
  - File descriptor of the socket,it is the value returned by socket( ) when socket was created.
  - P.TYPE:int
- *Sockaddr:*
  - This is a pointer.(contains the address of the server to which you want to connect to).
  - In practise this structure is typically a cast------> from a more specific structure depending upon the family or domain.EG.IPv4(AF_INET) would use 'sockaddr_in'.
  - P.TYPE: const struct sockaddr
- *Addrlen:*

    - it tells about the size of the address structure in bytes.
    - This typically tells about the sizeof(struct sockaddr_in).
    - P.TYPE:socklen_t
- Returns 0 on success and -1 on failure and errno.

C-RECAP*:const can be used to protect function parameters from being modified within the function, which is useful for ensuring that the function does not alter the input values.*

3.bind( )

- It is used to assign the local IP addr and port number to a socket and is used to bind the server to a particular address.
- Used in server side.
- It is typically used on the server side to specify the address on which the server will listen the incoming connection.
- **SYNTAX:** *int bind (int sockfd, const struct sockaddr *addr, socklen_t addrlen);*
- Same parameters as connect.
- Returns the same too

Note:
This use of the generic socket address sockaddr requires that any calls to these functions must cast the pointer to the protocol-specific address structure. For example for and IPv4 socket structure:

```
struct sockaddr_in serv; /* IPv4 socket address structure */
bind(sockfd, (struct sockaddr*) &serv, sizeof(serv));
```

why?
- When you call the bind() function, the **second parameter** is a pointer to a **struct sockaddr**. However, the "*actual structure you're working with is struct* ***sockaddr_in*** *for an IPv4 address*."
- The struct sockaddr_in is a protocol-specific structure that contains fields for the IP address and port number. The bind() function, however, is *designed to work with the more generic struct sockaddr*.
- *"Since* struct sockaddr_in is a specific structure for IPv4", -------> bind() expects a pointer to a generic struct sockaddr, you need to cast the pointer from struct sockaddr_in* to struct sockaddr*
- This casting tells the compiler to treat the pointer to the struct sockaddr_in as if it were a pointer to a struct sockaddr. This is necessary because the actual memory layout of the struct sockaddr_in is compatible with the expected struct sockaddr layout for the purposes of the bind() function.

4.listen( )

- Used in server side.
- To mark a socket as a *passive socket*,that is it will be used to accept the incoming requests.
- It prepares the socket to accept incoming connection requests.
- It converts an unconnected socket into passive socket,which means kernel will listen for an queue incoming requests.
- **SYNTAX:** *int listen (int sockfd, int backlog);*
- Wkt,*sockfd* is the file descriptor(fd) of the socket returned during socket( )
        - This is used to listen for the incoming connection requests.
- Backlog:

- It specifies max num of incoming connection that can be queued up for this socket,before the the system starts rejecting the incoming requests.
- It's the size size of the backlog queue.
- In other words,how many connections the system can hold before rejecting additional ones.

- If it's full,new connection attempts will be refused until space is in the queue.

Return type is 0 on success and -1 and 'errno' is set to indicate error.

5.accept( )

- Used on server side.
- It is used on server side to accept the incoming request from clients.
- *Def1:*The accept() function is used to accept an incoming connection request on a listening socket and create a new socket for communication with client.
- It retrieves the client's address information and return a anew socket descriptor for connection!
- *Def2:*It accepts the first conncetion from the backlog queue of pending connections,then creates a new socket for connection,and *returns a new fd(file discriptor)* that can be used for communication with the client.

- **SYNTAX:** *int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);*
- This *sockfd* is of the listening socket(that is of server).It is the same socket that was created with socket( ),bound with bound( ) and marked as passive with listen( ).
- *Struct sockaddr* will be filled with the address if the "connecting client"
- If client address is not known u can pass it as "NULL".
- *Addrlen*
  - This is a pointer to a "socklen_t varaible" that initially contains the address of the size of the addr buffer.
  - On return it contains the actual size of the client's address
  - P.TYPE:socklen_t
  - If addr is NULL ,then this parameter should also be NULL.

- Return value..function returns a fd ("connfd") on success which is used for actual communication with client
- On failure it return -1 and errno is set to indicate error.
- Because of creation of new socket specific to the connecting client,this allows server to handle multiple clinets simultaneously to handle multi clients each with it's own socket connection.
- It is a blocking call,that is it will wait till connection is available.

6.send( )

- Send( ) is used send data through a connected socket.
- **SYNTAX:** *ssize_t send(int sockfd, const void *buf, size_t len, int flags);*
- *Sockfd*
  - It is the fd though which data is **to** be sent.
  - It should be a socket that has been connected to a remote pear,thats is they typically established connect( ) on client side and accept( ) on server side.
- *Buf*
  - This is a pointer.

    - It is a pointer to the buffer containing the data you want to send.
    - This data in the buffer will be transmitted to the connected peer.
    - Buffer can hold any data including,string,binary and custom structures.
- Len
    - It tells how much data from the buffer to be sent.
    - The value is in bytes to be sent from buffer.
- Flags
    - This parameter allows us different option for sending data.
    - It is usually set to zero,but there are various options.
- Return on success------>the bytes actually sent,which can be < than len if network is congested.
- On failure return -1 and errno is set to indicate error.
- It can be used in blocking and non-blocking mode.

7.recv( )

- It is used to receive data **from** a connected socket.
- It's typically used on server side to receive data from client or on client side to receive data from server.

- **SYNTAX:** ssize_t recv(int sockfd, void *buf, size_t len, int flags);
- Sockdfd:the fd of the connected socket **from** which data is to be received.(from fd)
- Buf:
    - this is a pointer to the buffer where the received data will be stored.
    - This buffer must be large enough to hold data we expect to reieve.
- Len:
    - Maximum no of bytes to receive and store in the buffer.
    - It indicates the size of the buffer.
- Flags:
    - It allows options for controlling how the data is received.If not used,it is set to zero.
    - Return value return number of bytes actually received.
    - On failure returns -1 and errno is set to indicate error.

NOTE:*until now all the function are in the header <sys/socket.h>*

7.close( )

- This is in <unistd.h> header file.
- The fd is closed.
- It releases the resource associated with the socket.It is a standard I/O operation in UNIX-like system and is not only used for sockets but fd in genral.
- **SYNTAX:** int close(int sockfd);
- Sockfd:fd of the file or socket you want to close.

- The **bzero()** function is used to clear or zero out a block of memory. It is a legacy function in C, provided by the <string.h> header, and is used to set all bytes of a specified memory area to zero.
- ***SYNTAX:****void bzero(void *s, size_t len);*
- **s:** A pointer to the starting address of the memory block to be set to zero.
- **len:** The number of bytes to be set to zero.
- In the context of socket programming, bzero() is often used to initialize structures like struct sockaddr_in to ensure that all fields are cleared and set to zero.

- **INADDR_ANY**: This macro is used to allow the *server* to *bind to any available network interface*. It essentially means "listen on all available IP addresses."
- **htonl**: This function converts the *unsigned integer* INADDR_ANY from *host byte order to network byte order.* This is important because different systems may have different byte orders, and the network protocol expects a specific byte order.

```
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr.sin_port = htonl(PORT);
```

```
serveraddr.sin_family=AF_INET;
serveraddr.sin_addr.s_addr=htonl(INADDR_ANY);
serveraddr.sin_port=htonl(PORT);
```

htonl(): Converts a 32-bit (4-byte) value from host byte order to network byte order
htons():Converts a 16-bit (2-byte) value from host byte order to network byte order
inet_addr(): is a function in C used for converting an IPv4 address from its standard string representation into a numerical format that can be used in network programming.

UDP Server Side.

1. Create a UDP socket.*socket( )*
2. Bind the socket to a specific IP address and port.*bind( )*
3. Receive data from a client.*recvfrom( )*
4. Send the received data back to the client (echo it).*sendto( )*
5. Close the socket.*close( )*

UDP Client Side:

1. Create a UDP socket.*socket( )*
2. Send data to the server.*sendto( )*
3. Receive the echoed data from the server.*recvfrom( )*
4. Close the socket.*close( )*

8.sendto( )

- Function is used to send datat to a specific destination address in UDP.
- It describes the destination address each time it is sent.

- **SYNTAX:** ssize_t sendto (int sockfd , const void *buff, size_t len , int flags, const struct sockaddr *dest_addr,   socklen_t addrlen);

- Sockfd:
  - this is fd of the socket used for sending data.
  - This is obtained from socket()

- Buff:
    - It is a pointer to the buffer coantianig the data to be sent.
    - This is a constant so that it can't be altered during sending.
    - This holds the dtata you want to transmit to the destination.
- Len:
    - This tells the no of bytes to be sent from buffer.
    - This specifies the length of the message.
- Flag:
    - Tyoically set to zero.
    - But this can include option such as
    - MSG_DONTWAIT:for non-blocking mode.
    - MSG_CONFIRM:to conform the receipt
- Dest_addrelen:
    - It is a pointer that contains the destination address and port to which the data will be sent.
    - This allows specifying the recipient's address directly.
- Addrlen:
    - This describes the size of the size of the structure pointed to dest_addr.
    - This set to the size of the structure before call and will be used to send the correct length of the address.

9.recvfrom( )

- It is used in udp
- Used to receive message from a specific destination(IP addr and port).
- It allows to retrieve the address of the sender along with the data.
- **SYNTAX:** ssize_t recvfrom(int sockfd , void *buffer , size_t len ,int flags, struct sockaddr *src_addr , socklen_t * addrlen);

- Sockfd:
    - it is the fd of the socket from which data to be receieved.
- Buffer:
    - it is a pointer where the received data will be stored.
    - The size of buffer should be sufficient to hold the expected message.
- Len:
    - The maximum no of bytes to be read into the buffer.
    - This limits the sixe of the message that can be received in on call.
    - It indicates the size of the buffer.
- Flag:
    - It controls the behaviour of the function
    - it is typically set to zero.
- src_addr:
    - this is a pointer of struct sockaddr
    - this holds the address of the sender.
    - If u don't need to know the sender's address set it to NULL.

- Addrlen:
    - This initially contains the size of src_addr.
    - After the function call.it will be updated to aactual size of the address.
    - This can be set to NULL if the sender’s address is not needed.
- On success it returns the no of bytes received.
- If no data is received -1 is returned and errno is set to indicate error.