1)Connection Procedure followed in Client and Server Communication

In client-server communication, the connection procedure is designed to establish a network connection in a predictable and secure manner.

Server Initialization: The server initializes a socket and binds it to an IP address and port number where it will listen for incoming connection requests. For instance, an HTTP server usually binds to port 80.

Server Listening: The server invokes a listen() function on the socket which puts the server into listen mode where it waits for clients to connect. It's like a phone being turned on and waiting for calls.

Client Connection Request: When the client wants to connect to the server, it creates a socket and initiates a connection request to the server's IP address and port number. Imagine dialing a friend's number to start a phone call.

Handshake Process: Upon receiving a connection request, the server accepts the connection by performing a handshake process. In TCP, this is a three-way handshake involving SYN and ACK packets. It's akin to the server saying "Hello, who's this?" and the client responding, "It's me, can we talk?"

Establishing Connection: Once the handshake is successful, a connection is established. It's as if both the server and client agree to start a conversation over the phone.

Data Transfer: With the connection in place, the client and server can now exchange data. The server can respond to the client's requests, like answering a question over a call.

Closing Connection: After the exchange of data is completed, either the client or server can close the connection. It's like saying goodbye and hanging up the phone.

A real-time example would be accessing a website. When you type in a URL, your browser (client) requests a connection to the server where the website is hosted. The server, listening on port 80 (for HTTP) or port 443 (for HTTPS), accepts the connection. Your browser then requests the webpage content, which the server sends back for your browser to display. When you close the tab or move to another website, the connection is closed.

2)Bind() fucntion

The bind() function in socket programming is used to associate the socket with a local address, which consists of an IP address and a port number. This is necessary for servers because they must operate on a specific port so that clients know where to connect to. The bind() function takes a socket descriptor and an address as parameters, and it tells the operating system that the specified socket will be used to accept incoming connections or data on that address.

If the port is already in use or the program doesn't have the privilege to bind to the specified port (typically ports below 1024 require administrative privileges), the bind() call will fail, and an exception will be thrown.

3)

A Datagram Socket is used for sending and receiving datagram packets, which are self-contained messages that are sent without establishing a connection between the sender and receiver. This is known as connectionless communication and is provided by the User Datagram Protocol (UDP).

In datagram communication, each packet (datagram) is an independent entity containing enough information to be routed from the source to the destination without relying on exchanges between the sender, receiver, and the transport infrastructure. Because there is no need to establish a connection, datagram sockets can be used to send messages to multiple recipients, and they can receive messages from any sender.

Explanation in a Real-Time Scenario:

The server code sets up a UDP server that can receive messages from any client. It binds to a specific port (9999 in this case) on all interfaces on the server machine. Once bound, the server enters an infinite loop, waiting for messages with recvfrom(). When a message arrives, the server prints it out, then echoes the message back to the sender using sendto().

The client code, intended to run on a client machine, creates a UDP socket. It sends a message to the server using sendto(), which requires the server's address and port number. After sending the message, it waits for a reply with recvfrom(). Upon receiving the reply, the client prints it and closes its socket.

4)

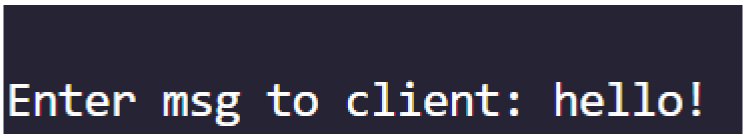
Server:

Code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<string.h>

int main(){
int socketmain, socketclient, port=5000;
struct sockaddr_in serv, clientaddr;
socklen_t clientlen;

if((socketmain=socket(AF_INET, SOCK_STREAM, 0))<0){
printf("\nServer cant open socket");
exit(0);
}

bzero(&serv, sizeof(serv));
serv.sin_family=AF_INET;
serv.sin_addr.s_addr=htonl(INADDR_ANY);
serv.sin_port=htons(port);

if((bind(socketmain,(struct sockaddr*)&serv, sizeof(serv)))<0){
printf("\nserver bind failed");
exit(0);
}

listen(socketmain,5);

if((socketclient=accept(socketmain,(struct sockaddr*)&clientaddr, &clientlen))<0){
printf("\nclient is bad");
exit(0);
}
```

```c
char buf[100];
printf("\nEnter msg to client: ");
scanf("%s",buf);
write(socketclient,buf,100);
close(socketmain);
return 0;
}
```

Output:



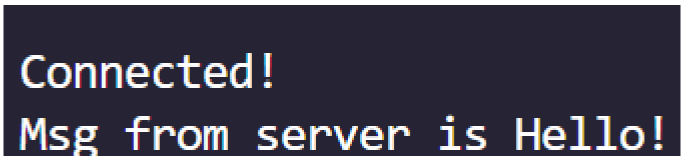
```
Enter msg to client: hello!
```

Client:

Code:

```c
#include<stdio.h>
#include<sys/types.h>
#include<winsock2.h>
//#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<string.h>

int main(){
int sockfd, n, port=5000;
struct sockaddr_in serv;
char buf[100];

if((sockfd=socket(AF_INET, SOCK_STREAM, 0))<0){
printf("\nSocket cant be opened ");
exit(0);
}

bzero(&serv, sizeof(serv));
serv.sin_family=AF_INET;
serv.sin_port=htons(port);

if((connect(sockfd,(struct sockaddr*)&serv, sizeof(serv)))<0){
printf("\nConnection failed");
exit(0);
}

printf("\nConnected!");

read(sockfd,buf,100);
printf("\nMsg from Server is %s\n", buf);
close(sockfd);
return 0;
}
```

Output:



```
Connected!
Msg from server is Hello!
```

5)

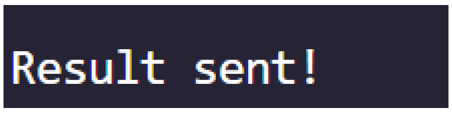
Server:

Code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<string.h>

int main(){
int socketmain, socketclient, port=5000,i;
struct sockaddr_in serv, clientaddr;
socklen_t clientlen;

if((socketmain=socket(AF_INET, SOCK_STREAM, 0))<0){
printf("\nServer cant open socket");
exit(0);
}

bzero(&serv, sizeof(serv));
serv.sin_family=AF_INET;
serv.sin_addr.s_addr=htonl(INADDR_ANY);
serv.sin_port=htons(port);

if((bind(socketmain,(struct sockaddr*)&serv, sizeof(serv)))<0){
printf("\nserver bind failed");
exit(0);
}

listen(socketmain,5);

if((socketclient=accept(socketmain,(struct sockaddr*)&clientaddr, &clientlen))<0){
printf("\nclient is bad");
exit(0);
```

```c
}

char buf[100],str[100];
read(socketclient,buf,100);
for(i=0;i<strlen(buf);i++){
    str[i]=buf[strlen(buf)-1-i];
}
if(strcmp(buf,str)==0)
    strcpy(buf,"Palindrome");
else
    strcpy(buf,"Not a Palindrome");
write(socketclient,buf,100);
printf("Result sent!\n");
close(socketmain);
return 0;
}
```

Output:



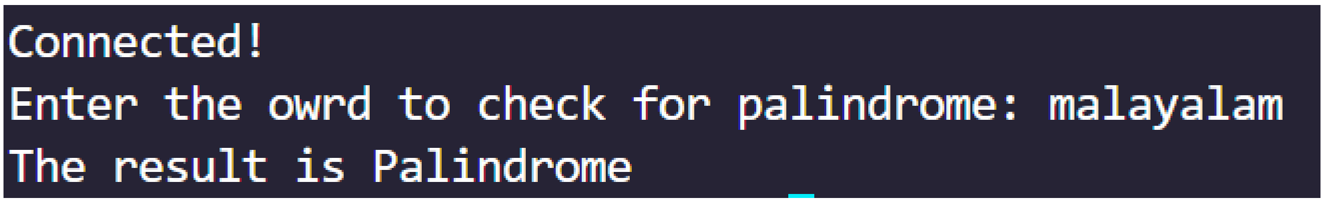
```
Result sent!
```

Client:

Code:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<string.h>

int main(){
int sockfd, n, port=5000;
struct sockaddr_in serv;
char buf[100];

if((sockfd=socket(AF_INET, SOCK_STREAM, 0))<0){
printf("\nSocket cant be opened ");
exit(0);
}

bzero(&serv, sizeof(serv));
serv.sin_family=AF_INET;
serv.sin_port=htons(port);

if((connect(sockfd,(struct sockaddr*)&serv, sizeof(serv)))<0){
printf("\nConnection failed");
exit(0);
}
printf("\nConnected!");

printf("\nEnter the word to check for palindrome: ");
scanf("%s",buf);
write(sockfd,buf,100);
read(sockfd,buf,100);
printf("The result is %s\n", buf);
close(sockfd);
return 0;
}
```

Output:



```
Connected!
Enter the owrd to check for palindrome: malayalam
The result is Palindrome
```

6)

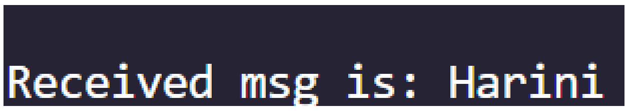
Server:

Code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<string.h>

int main(){
int fd, socketclient, port=5000,i;
struct sockaddr_in serv, clientaddr;
socklen_t clientlen;
char buf[100];

if((fd=socket(AF_INET, SOCK_DGRAM, 0))<0){
printf("\nServer cant open socket");
exit(0);
}

bzero(&serv, sizeof(serv));
serv.sin_family=AF_INET;
serv.sin_addr.s_addr=htonl(INADDR_ANY);
serv.sin_port=htons(port);
```

```c
if((bind(fd,(struct sockaddr*)&serv, sizeof(serv)))<0){
printf("\nserver bind failed");
exit(0);
}

int l=sizeof(clientaddr);
recvfrom(fd,buf,sizeof(buf),0,(struct sockaddr*)&clientaddr, &l);
printf("\nReceived msg is: %s\n", buf);
close(fd);
return 0;
}
```

Output:



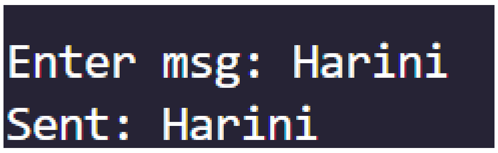
```
Received msg is: Harini
```

Client:

Code:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<string.h>

int main(){
int sockfd, port=5000;
struct sockaddr_in serv;
char buf[100];

if((sockfd=socket(AF_INET, SOCK_DGRAM, 0))<0){
printf("\nSocket cant be opened ");
exit(0);
}

bzero(&serv, sizeof(serv));
serv.sin_family=AF_INET;
serv.sin_port=htons(port);

printf("\nEnter msg: ");
scanf("%s",buf);

sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr*)&serv, sizeof(serv));

printf("Sent: %s\n",buf);
close(sockfd);
return 0;
}
```

Output:



```
Enter msg: Harini
Sent: Harini
```