

## MODULE-4

### 1. Explain the connection procedure followed in client server communication

Using sockets enables the initiation of new communication channels, facilitating data exchange over a network. Binding associates a local address with the socket, establishing its connection point. Listening allows for the acceptance of incoming connections, while the accept function blocks until a request is received. Connecting initiates an attempt to establish a connection with another endpoint. Sending data allows for the transmission of information across the network, while receiving enables the retrieval of data from remote sources. Finally, closing terminates the connection, releasing network resources. These functions collectively form the basis for establishing, maintaining, and terminating network communications using sockets.

- a) **Socket:** With the help of a socket, we can create a new communication.
- b) **Bind:** With the help of this we can, we can attach the local address with the socket.
- c) **Listen:** With this help; we can accept the connection.
- d) **Accept:** With this help; we can block the incoming connection until the request arrives.
- e) **Connect:** With this help; we can attempt to establish the connection.
- f) **Send:** With the help of this; we can send the data over the network.
- g) **Receive:** With this help; we can receive the data over the network.
- h) **Close:** With the help of this, we can release the connection from the network.

### 2. What is the use of bind() function in socket programming ?

In socket programming, the bind() function is used to associate a local address, typically an IP address and a port number, with a socket. This binding operation is crucial for the socket to communicate with other sockets over a network. By binding a socket to a specific local address, the operating system knows which network interface and port to use for sending and receiving data. Without binding, the OS would assign a random port number and possibly an IP address to the socket, making it difficult for other sockets to know where to connect or send data. Therefore, the bind() function is essential for establishing a well-defined communication endpoint for the socket within the network environment.

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

### 3. What is Datagram Socket ?

A datagram socket is a type of network communication mechanism that operates on the User Datagram Protocol (UDP). Unlike TCP Sockets, which provide reliable, ordered, and connection-oriented communication (such as with TCP), datagram sockets offer a connection-less communication model.

In a datagram socket, data is sent and received in discrete units known as datagrams. Each datagram is independent of the others, meaning there is no guarantee of delivery, and they may arrive out of order or not at all.

**4. Write a server/client model socket program to exchange hello message between them.**

**Server:**

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

int main(){

    char *ip = "127.0.0.1";

    int port = 5200;

    int server_sock, client_sock;

    struct sockaddr_in server_addr, client_addr;

    socklen_t addr_size;

    char buffer[1024];

    int n;

    server_sock = socket(AF_INET, SOCK_STREAM, 0);

    if (server_sock < 0){

        perror("[-]Socket error");

        exit(1);

    }

    printf("[+]TCP server socket created.\n");

    memset(&server_addr, '\0', sizeof(server_addr));

    server_addr.sin_family = AF_INET;
```

```

server_addr.sin_port = port;

server_addr.sin_addr.s_addr = inet_addr(ip);

n = bind(server_sock, (struct sockaddr*)&server_addr, sizeof(server_addr));

if (n < 0){

    perror("[-]Bind error");

    exit(1);

}

printf("[+]Bind to the port number: %d\n", port);

listen(server_sock, 5);

printf("Listening...\n");

while(1){

    addr_size = sizeof(client_addr);

    client_sock = accept(server_sock, (struct sockaddr*)&client_addr, &addr_size);

    printf("[+]Client connected.\n");

    bzero(buffer, 1024);

    recv(client_sock, buffer, sizeof(buffer), 0);

    printf("Client: %s\n", buffer);

    bzero(buffer, 1024);

    strcpy(buffer, "Hello, I am Server");

    send(client_sock, buffer, strlen(buffer), 0);

    close(client_sock);

    printf("[+]Client disconnected.\n\n");

}

return 0;

}

```

**Client:**

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

int main(){

    char *ip = "127.0.0.1";

    int port = 5200;

    int sock;

    struct sockaddr_in addr;

    socklen_t addr_size;

    char buffer[1024];

    int n;

    sock = socket(AF_INET, SOCK_STREAM, 0);

    if (sock < 0){

        perror("[-]Socket error");

        exit(1);

    }

    printf("[+]TCP server socket created.\n");


    memset(&addr, '\0', sizeof(addr));

    addr.sin_family = AF_INET;

    addr.sin_port = port;

    addr.sin_addr.s_addr = inet_addr(ip);
```

```

connect(sock, (struct sockaddr*)&addr, sizeof(addr));

printf("Connected to the server.\n");

bzero(buffer, 1024);

strcpy(buffer, "Hello, I am Client.");

send(sock, buffer, strlen(buffer), 0);

bzero(buffer, 1024);

recv(sock, buffer, sizeof(buffer), 0);

printf("Server: %s\n", buffer);

close(sock);

printf("Disconnected from the server.\n");

return 0;

}

```

Output:

```

client.c - TCP - Visual Studio Code

Selection View Go Run Terminal Help

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

manoj@manoj-virtual-machine:~/Desktop/Socket_Programming/TCP$ ./server
[+]TCP server socket created.
[+]Bind to the port number: 5200
listening...
[+]Client connected.
Client: Hello, I am Client.
[+]Client disconnected.

manoj@manoj-virtual-machine:~/Desktop/Socket_Programming/TCP$ ./client
[+]TCP server socket created.
Connected to the server.
Server: Hello, I am Server
Disconnected from the server.
manoj@manoj-virtual-machine:~/Desktop/Socket_Programming/TCP$

```

**5. Write a TCP server-client program to check if a given string is Palindrome**

**Input: level**

**Output: Palindrome**

**Input: Assessment**

**Output: Not a Palindrome**

**Server:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(){

    char *ip = "127.0.0.1";
    int port = 5200;

    int server_sock, client_sock;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    char buffer[1024];
    int n;

    server_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (server_sock < 0){
        perror("[-]Socket error");
        exit(1);
    }
    printf("[+]TCP server socket created.\n");

    memset(&server_addr, '\0', sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = port;
    server_addr.sin_addr.s_addr = inet_addr(ip);
    n=bind(server_sock,(structsockaddr*)&server_addr,sizeof(server_addr));
```

```

if (n < 0){
    perror("[-]Bind error");
    exit(1);
}
printf("[+]Bind to the port number: %d\n", port);

listen(server_sock, 5);
printf("Listening...\n");

while(1){
    addr_size = sizeof(client_addr);
    client_sock = accept(server_sock, (struct sockaddr*)&client_addr,
&addr_size);
    printf("[+]Client connected.\n");

    bzero(buffer, 1024);
    recv(client_sock, buffer, sizeof(buffer), 0);
    printf("Client: %s\n",buffer);
    int l = 0;
    int h = strlen(buffer) - 1;
    while (h > l) {
        if (buffer[l++] != buffer[h--]) {
            bzero(buffer, 1024);
            strcpy(buffer, "The given string is not Palindrome");
            send(client_sock, buffer, strlen(buffer), 0);
            exit(1);
        }
    }
    bzero(buffer, 1024);
    strcpy(buffer, "The given string is Palindrome");
    send(client_sock, buffer, strlen(buffer), 0);

    close(client_sock);
    printf("[+]Client disconnected.\n\n");

}

return 0;
}

```

**Client:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(){

    char *ip = "127.0.0.1";
    int port = 5200;

    int sock;
    struct sockaddr_in addr;
    socklen_t addr_size;
    char buffer[1024];
    int n;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0){
        perror("[-]Socket error");
        exit(1);
    }
    printf("[+]TCP server socket created.\n");

    memset(&addr, '\0', sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = port;
    addr.sin_addr.s_addr = inet_addr(ip);

    connect(sock, (struct sockaddr*)&addr, sizeof(addr));
    printf("Connected to the server.\n");

    bzero(buffer, 1024);
    printf("Enter a string:");
    scanf("%s",buffer);
    send(sock, buffer, strlen(buffer), 0);
    bzero(buffer, 1024);
    recv(sock, buffer, sizeof(buffer), 0);
    printf("Server: %s\n", buffer);

    close(sock);
    printf("Disconnected from the server.\n");
```



```

return 0;

}

```

Output:

```

client.c - Palindrome - Visual Studio Code

File Edit Selection View Go Run Terminal Help

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

manoj@manoj-virtual-machine:~/Desktop/Palindrome$ ./serverp
[+]TCP server socket created.
[+]Bind to the port number: 5200
Listening...
[+]Client connected.
Client: level
[+]Client disconnected.

[+]Client connected.
Client: assessment
manoj@manoj-virtual-machine:~/Desktop/Palindrome$

manoj@manoj-virtual-machine:~/Desktop/Palindrome$ ./clientp
[+]TCP server socket created.
Connected to the server.
Enter a string:level
Server: The given string is Palindrome
Disconnected from the server.

manoj@manoj-virtual-machine:~/Desktop/Palindrome$ ./clientp
[+]TCP server socket created.
Connected to the server.
Enter a string:assessment
Server: The given string is not Palindrome
Disconnected from the server.
manoj@manoj-virtual-machine:~/Desktop/Palindrome$

```

## 6. Write an example to demonstrate UDP server-client program

**Server:**

```

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

int main(int argc, char **argv){

    if (argc != 2){
        printf("Usage: %s <port>\n", argv[0]);
        exit(0);
    }

    char *ip = "127.0.0.1";
    int port = atoi(argv[1]);

```

```

int sockfd;
struct sockaddr_in server_addr, client_addr;
char buffer[1024];
socklen_t addr_size;
int n;

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0){
    perror("[-]socket error");
    exit(1);
}

memset(&server_addr, '\0', sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
server_addr.sin_addr.s_addr = inet_addr(ip);

n = bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr));
if (n < 0) {
    perror("[-]bind error");
    exit(1);
}

bzero(buffer, 1024);
addr_size = sizeof(client_addr);
recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr*)&client_addr, &addr_size);
printf("[+]Data recv: %s\n", buffer);

bzero(buffer, 1024);
strcpy(buffer, "Welcome to the UDP Server.");
sendto(sockfd, buffer, 1024, 0, (struct sockaddr*)&client_addr, sizeof(client_addr));
printf("[+]Data send: %s\n", buffer);

return 0;
}

```

### **Client:**

```

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

int main(int argc, char **argv){

    if (argc != 2) {
        printf("Usage: %s <port>\n", argv[0]);
    }
}

```

```

    exit(0);
}

char *ip = "127.0.0.1";
int port = atoi(argv[1]);

int sockfd;
struct sockaddr_in addr;
char buffer[1024];
socklen_t addr_size;

sockfd = socket(AF_INET, SOCK_DGRAM, 0);
memset(&addr, '\0', sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr.s_addr = inet_addr(ip);

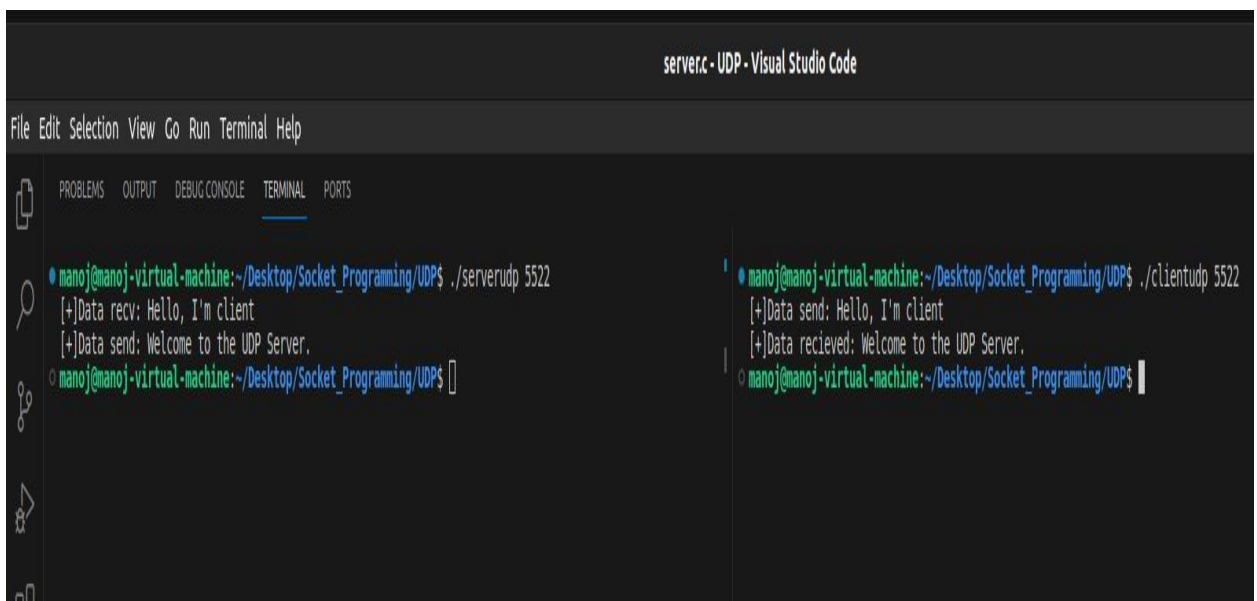
bzero(buffer, 1024);
strcpy(buffer, "Hello, I'm client");
sendto(sockfd, buffer, 1024, 0, (struct sockaddr*)&addr, sizeof(addr));
printf("[+]Data send: %s\n", buffer);

bzero(buffer, 1024);
addr_size = sizeof(addr);
recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr*)&addr, &addr_size);
printf("[+]Data recieved: %s\n", buffer);

return 0;
}

```

## Output:



```

server.c - UDP - Visual Studio Code

File Edit Selection View Go Run Terminal Help

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

manoj@manoj-virtual-machine:~/Desktop/Socket_Programming/UDP$ ./serverudp 5522
[+]Data recv: Hello, I'm client
[+]Data send: Welcome to the UDP Server.
manoj@manoj-virtual-machine:~/Desktop/Socket_Programming/UDP$

manoj@manoj-virtual-machine:~/Desktop/Socket_Programming/UDP$ ./clientudp 5522
[+]Data send: Hello, I'm client
[+]Data recieved: Welcome to the UDP Server.
manoj@manoj-virtual-machine:~/Desktop/Socket_Programming/UDP$

```