ADVANCED C PROGRAMMING – MODULE 4

1. **Explain the connection procedure followed in client server communication**
   In an Operating System, Client Server Communication refers to the exchange of data and Services among multiple machines or processes.
   In Client Server Communication we can use different ways.
   a. Sockets Mechanism
   b. Remote Procedure Call
   c. Message Passing
   d. Inter-process Communication
   e. Distributed File Systems

   But in general the connection procedure in client-server communication typically follows several steps:

   1. Server Initialization
   2. Client Request
   3. Server Listening
   4. Handshake
   5. Connection Establishment
   6. Data Exchange
   7. Connection Termination

2. **What is the use of bind() function in socket programming?**
   In socket programming, the bind() function is used to associate a specific network address (IP address) and port number with a socket. This binding operation is essential for servers, as it allows them to specify the network interface and port on which they will listen for incoming connections.

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

3. **What is Datagram Socket?**
   A datagram socket is a type of socket used in networking that provides a connectionless, unreliable communication mechanism between two endpoints. Datagram sockets are commonly associated with the User Datagram Protocol (UDP), although they can be used with other protocols as well.

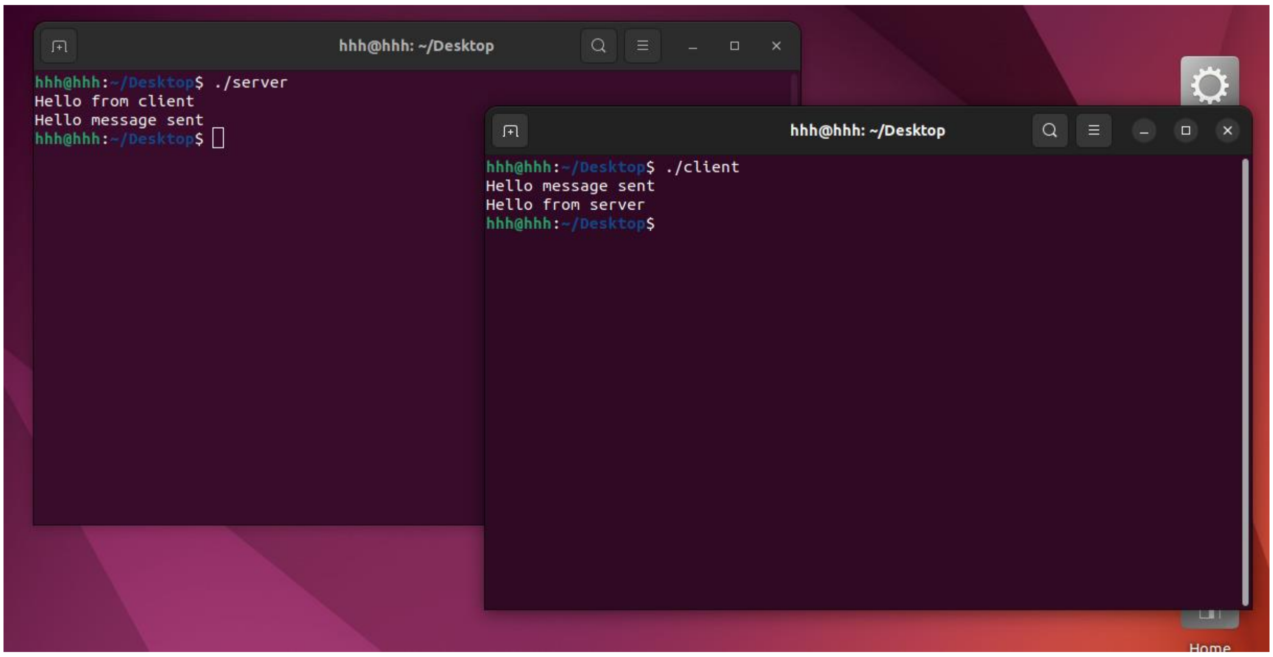
4. **Write a server/client model socket program to exchange hello message between them.**

   //server
```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080
int main(int argc, char const* argv[])
{
        int server_fd, new_socket;
        ssize_t valread;
        struct sockaddr_in address;
        int opt = 1;
        socklen_t addrlen = sizeof(address);
```

```
	char buffer[1024] = { 0 };
	char* hello = "Hello from server";

	// Creating socket file descriptor
	if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
		perror("socket failed");
		exit(EXIT_FAILURE);
	}

	// Forcefully attaching socket to the port 8080
	if (setsockopt(server_fd, SOL_SOCKET,
				SO_REUSEADDR | SO_REUSEPORT, &opt,
				sizeof(opt))) {
		perror("setsockopt");
		exit(EXIT_FAILURE);
	}
	address.sin_family = AF_INET;
	address.sin_addr.s_addr = INADDR_ANY;
	address.sin_port = htons(PORT);

	// Forcefully attaching socket to the port 8080
	if (bind(server_fd, (struct sockaddr*)&address,
			sizeof(address))
		< 0) {
		perror("bind failed");
		exit(EXIT_FAILURE);
	}
	if (listen(server_fd, 3) < 0) {
		perror("listen");
		exit(EXIT_FAILURE);
	}
	if ((new_socket
		= accept(server_fd, (struct sockaddr*)&address,
				&addrlen))
		< 0) {
		perror("accept");
		exit(EXIT_FAILURE);
	}
	valread = read(new_socket, buffer,1024 - 1); // subtract 1 for the null
	printf("%s\n", buffer);
	send(new_socket, hello, strlen(hello), 0);
	printf("Hello message sent\n");

	// closing the connected socket
	close(new_socket);
	// closing the listening socket
	close(server_fd);
	return 0;
}
```

//client

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080

int main(int argc, char const* argv[])
{
        int status, valread, client_fd;
        struct sockaddr_in serv_addr;
        char* hello = "Hello from client";
        char buffer[1024] = { 0 };
        if ((client_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                printf("\n Socket creation error \n");
                return -1;
        }

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(PORT);

        // Convert IPv4 and IPv6 addresses from text to binary
        // form
        if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
                <= 0) {
                printf(
                        "\nInvalid address/ Address not supported \n");
                return -1;
        }

        if ((status
                = connect(client_fd, (struct sockaddr*)&serv_addr,
                                sizeof(serv_addr)))
                < 0) {
                printf("\nConnection Failed \n");
                return -1;
        }
        send(client_fd, hello, strlen(hello), 0);
        printf("Hello message sent\n");
        valread = read(client_fd, buffer,1024 - 1); // subtract 1 for the null
        printf("%s\n", buffer);

        // closing the connected socket
        close(client_fd);
        return 0;
}
```



```
hhh@hhh:~/Desktop$ ./server
Hello from client
Hello message sent
hhh@hhh:~/Desktop$
```

```
hhh@hhh:~/Desktop$ ./client
Hello message sent
Hello from server
hhh@hhh:~/Desktop$
```

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

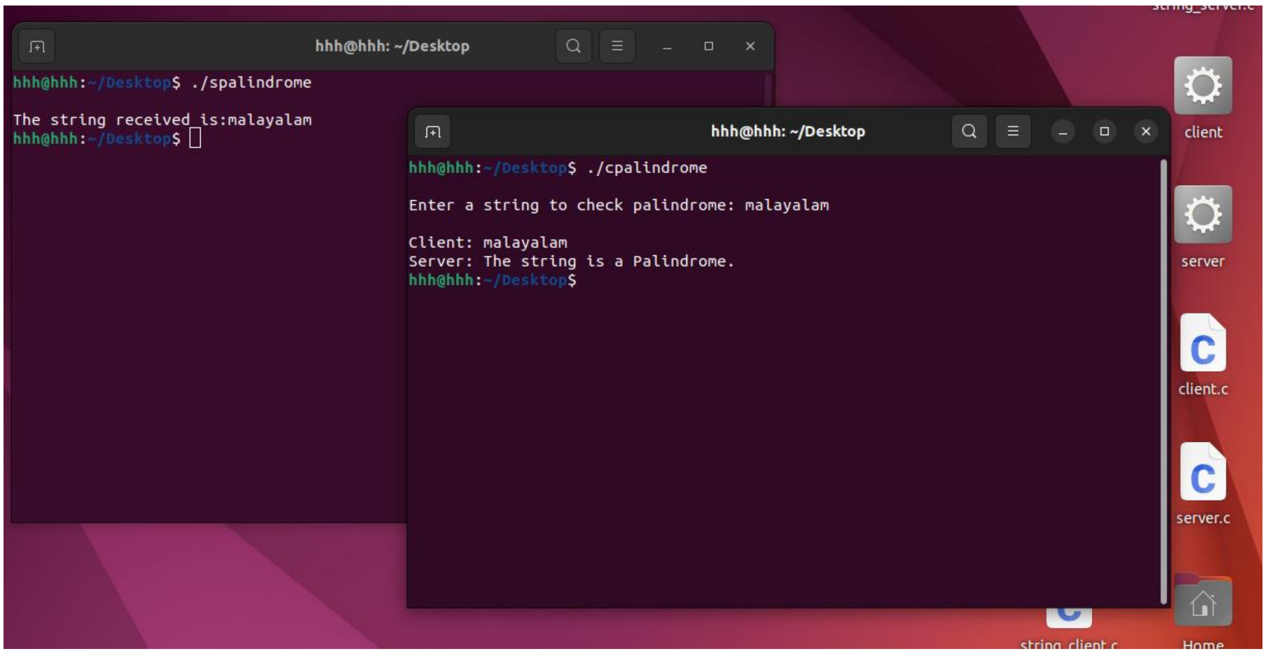
```
//server
// defines in_addr structure
#include <arpa/inet.h>

// contains constants and structures
// needed for internet domain addresses
#include <netinet/in.h>

// standard input and output library
#include <stdio.h>

// contains string functions
#include <string.h>

// for socket creation
#include <sys/socket.h>

// contains constructs that facilitate getting
// information about files attributes.
#include <sys/stat.h>

// contains a number of basic derived types
// that should be used whenever appropriate
#include <sys/types.h>

int main()
{
        struct sockaddr_in client, server;
        int s, n, sock, g, j, left, right, flag;
        char b1[20], b2[10], b3[10], b4[10];
```

```
	// creating socket
	s = socket(AF_INET, SOCK_STREAM, 0);

	// assign IP, PORT
	server.sin_family = AF_INET;

	// this is the port number of running server
	server.sin_port = 2000;
	server.sin_addr.s_addr = inet_addr("127.0.0.1");

	// Binding newly created socket
	// to given IP and verification
	bind(s, (struct sockaddr*)&server, sizeof server);
	listen(s, 1);
	n = sizeof client;

	sock = accept(s, (struct sockaddr*)&client, &n);
	for (;;) {
		recv(sock, b1, sizeof(b1), 0);

		// whenever a request from a client came.
		// It will be processed here.
		printf("\nThe string received is:%s\n", b1);
		if (strlen(b1) == 0)
			flag = 1;
		else {
			left = 0;
			right = strlen(b1) - 1;
			flag = 1;
			while (left < right && flag) {
				if (b1[left] != b1[right])
					flag = 0;
				else {
					left++;
					right--;
				}
			}
		}
		send(sock, &flag, sizeof(int), 0);
		break;
	}
	close(sock);

	// close the socket
	close(s);
}
```

//client

```
#include <arpa/inet.h>
```

```
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
        struct sockaddr_in client;
        int s, flag;
        char buffer[20];

        // socket create
        s = socket(AF_INET, SOCK_STREAM, 0);

        // assign IP, PORT
        client.sin_family = AF_INET;
        client.sin_port = 2000;
        client.sin_addr.s_addr = inet_addr("127.0.0.1");

        // connect the client socket to server socket
        connect(s, (struct sockaddr*)&client, sizeof client);

        for (;;) {
                printf("\nEnter a string to check palindrome: ");
                scanf("%s", buffer);

                printf("\nClient: %s", buffer);
                send(s, buffer, sizeof(buffer), 0);
                recv(s, &flag, sizeof(int), 0);

                if (flag == 1) {
                        printf("\nServer: The string is a Palindrome.\n");
                        break;
                }
                else {
                        printf("\nServer: The string is not a palindrome.\n");
                        break;
                }
        }

        // close the socket
        close(s);
}
```



```
hhh@hhh:~/Desktop$ ./spalindrome

The string received is:malayalam
hhh@hhh:~/Desktop$ 
```

```
hhh@hhh:~/Desktop$ ./cpalindrome

Enter a string to check palindrome: malayalam

Client: malayalam
Server: The string is a Palindrome.
hhh@hhh:~/Desktop$
```

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

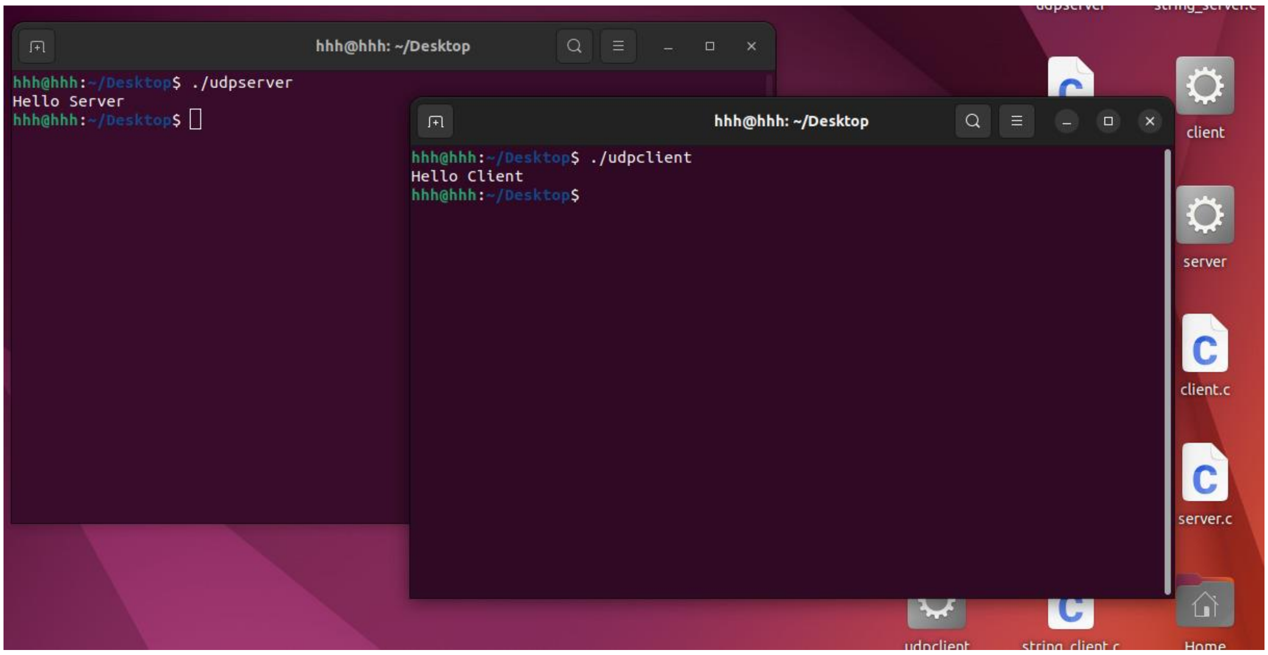
```
//server
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include<netinet/in.h>
#define PORT 5000
#define MAXLINE 1000

// Driver code
int main()
{
        char buffer[100];
        char *message = "Hello Client";
        int listenfd, len;
        struct sockaddr_in servaddr, cliaddr;
        bzero(&servaddr, sizeof(servaddr));

        // Create a UDP Socket
        listenfd = socket(AF_INET, SOCK_DGRAM, 0);
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(PORT);
        servaddr.sin_family = AF_INET;

        // bind server address to socket descriptor
        bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

        //receive the datagram
        len = sizeof(cliaddr);
```

```
	int n = recvfrom(listenfd, buffer, sizeof(buffer),
			0, (struct sockaddr*)&cliaddr,&len); //receive message from server
	buffer[n] = '\0';
	puts(buffer);

	// send the response
	sendto(listenfd, message, MAXLINE, 0,
		(struct sockaddr*)&cliaddr, sizeof(cliaddr));
}
```

//client

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

#define PORT 5000
#define MAXLINE 1000

// Driver code
int main()
{
	char buffer[100];
	char *message = "Hello Server";
	int sockfd, n;
	struct sockaddr_in servaddr;

	// clear servaddr
	bzero(&servaddr, sizeof(servaddr));
	servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	servaddr.sin_port = htons(PORT);
	servaddr.sin_family = AF_INET;

	// create datagram socket
	sockfd = socket(AF_INET, SOCK_DGRAM, 0);

	// connect to server
	if(connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
	{
		printf("\n Error : Connect Failed \n");
		exit(0);
	}

	// request to send datagram
	// no need to specify server address in sendto
```

```
        // connect stores the peers IP and port
        sendto(sockfd, message, MAXLINE, 0, (struct sockaddr*)NULL, sizeof(servaddr));

        // waiting for response
        recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)NULL, NULL);
        puts(buffer);

        // close the descriptor
        close(sockfd);
}
```



```
hhh@hhh:~/Desktop$ ./udpserver
Hello Server
hhh@hhh:~/Desktop$ 
```

```
hhh@hhh:~/Desktop$ ./udpclient
Hello Client
hhh@hhh:~/Desktop$
```