Module 4 – Socket Programming

1. **Explain the connection procedure followed in client-server communication**

- In client-server communication, the connection procedure typically follows these steps:
- The server initializes itself by creating a socket using the `socket()` system call. The socket is bound to a specific port and address using `bind()` to allow clients to connect to it.
- After binding to an address and port, the server listens for incoming client connections using the `listen()` system call. This allows the server to accept incoming connection requests from clients.
- The client initializes itself by creating a socket using the `socket()` system call.
- The client connects to the server by specifying the server's address and port using the `connect()` system call. This establishes a connection to the server's socket, allowing communication between the client and server.
- For TCP servers, once the server socket is listening, it accepts incoming connection requests from clients using the `accept()` system call. This creates a new socket for communication with the client while the server socket continues to listen for new connections.
- After the connection is established, both the client and server can send and receive data through their respective sockets using the `send()`, `recv()` (for TCP) or `sendto()`, `recvfrom()` (for UDP) system calls.
- Once communication is complete or terminated by either the client or server, they close their sockets using the `close()` system call to release the resources associated with the connection.
- This connection procedure ensures that clients can establish a connection to the server, allowing them to exchange data in a reliable and organized manner. Depending on the protocol (TCP or UDP), the connection establishment process may vary slightly, especially in how connections are initiated and maintained.

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

- First, we create a socket, which is like a communication endpoint.
- Once we have the socket, we need to tell the system which address and port we want to use for communication. The bind() function is used for this.
- When we use INADDR_ANY, it means we're allowing connections from any available network interface on the server.
- Hence, the bind() function connects our socket to a specific address and port for communication, and using INADDR_ANY means we're open to connections from any available network interface.
- It is created as follows

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

3. **What is Datagram Socket ?**

- Datagram sockets provide a way for processes to communicate using the User Datagram Protocol (UDP). They offer a simple and connectionless communication method.
- Datagram sockets support a bidirectional flow of messages. This means both sending and receiving messages is possible using the same socket.
- Unlike other protocols like TCP, UDP does not guarantee the ordering of messages. In the context of datagram sockets, this means that a process may receive messages in a different order from the order in which they were sent.
- Due to the unreliable nature of UDP, it's possible for datagram sockets to receive duplicate messages. This could occur if a message is sent multiple times and some of the duplicates arrive at the destination.
- Datagram sockets preserve record boundaries in the data. This means that if a message is sent in distinct chunks or records, those boundaries will be maintained when the message is received.
- Datagram sockets are created with the `SOCK_DGRAM` type. This indicates to the system that the socket will be used for UDP communication.
- In summary, datagram sockets using UDP provide a lightweight, connectionless communication method where message ordering is not guaranteed, duplicate messages may occur, and record boundaries are preserved. They are suitable for scenarios where strict reliability and ordering are not critical, such as real-time multimedia streaming or simple request-response protocols.

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

C Program to Create Server

```
#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);
  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);
  printf("%s\n", buffer);
  close(client_fd);
  return 0;
}
```

C Program to Create Client

```
#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";

  if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
```

```
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    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);

    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);
    printf("%s\n", buffer);
    send(new_socket, hello, strlen(hello), 0);
    printf("Hello message sent\n");

    close(new_socket);
    close(server_fd);
    return 0;
}
```

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

C Program to Create Server

```
#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, server;
   int s, n, sock, g, j, left, right, flag;
   char b1[20], b2[10], b3[10], b4[10];

   s = socket(AF_INET, SOCK_STREAM, 0);

   server.sin_family = AF_INET;
   server.sin_port = 2000;
   server.sin_addr.s_addr = inet_addr("127.0.0.1");

   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);
      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(s);
  return 0;
}
```

C Program to Create 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];

   s = socket(AF_INET, SOCK_STREAM, 0);

   client.sin_family = AF_INET;
   client.sin_port = 2000;
   client.sin_addr.s_addr = inet_addr("127.0.0.1");

   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(s);
}
```

OUTPUT:

```
Enter a string to check palindrome: assessment

Client: assessment
Server: The string is not a palindrome.


=== Code Execution Successful ===
```

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

UDP SERVER CODE

```
#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;
}
```

UDP CLIENT CODE

```
#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, World!");
  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 recv: %s\n", buffer);

  return 0;
}
```