MODULE 4 ASSIGNMNETS

1. Procedures of client-server communication.

   Server Initialization:

   The server initializes and binds a socket to a specific address and port and listens for incoming connection requests from clients on this socket.

   Client Initialization:

   The client creates a socket and specifies the address and port of the server it wants to connect to.

   Connection Establishment:

   - The client initiates a connection request to the server by sending a SYN signal.
   - The server responds with a SYN-ACK signal.
   - The client acknowledges the server's response by sending an ACK signal.

   Data Transfer:

   After the connection is established, the client and server can exchange data using read and write operations on their sockets.

   Connection Termination:

   The client or server can initiate the termination of the connection when communication is complete.

   - The client or server sends a FIN signal to indicate its intention to close the connection.
   - The other responds with an ACK signal.
   - Finally, both send FIN signal to acknowledge the termination of the connection.

2. Use of bind().

   Bind() is used in socket programming to assign an ip address and port number for the socket created beforehand which enables the client to identify and enables communication.

3. Datagram socket.

   A socket that uses UDP as transmission protocol is termed as datagram socket this enables connection-less communication and async transmission of data between client and the server.

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

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

#define PORT 8080
#define MAX_MESSAGE_SIZE 100

int main() {
   int server_fd, new_socket;
   struct sockaddr_in address;
   int addrlen = sizeof(address);
   char buffer[MAX_MESSAGE_SIZE] = {0};
   const char *hello_message = "Hello from server";
   const char *client_hello_message = "Hello from client";

   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) 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)) == -1)
exit(EXIT_FAILURE);
   if (listen(server_fd, 3) == -1) exit(EXIT_FAILURE);
   if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t
*)&addrlen)) == -1) exit(EXIT_FAILURE);
   send(new_socket, hello_message, strlen(hello_message), 0);
   printf("Hello message sent to client\n");

   int sock = 0;
   struct sockaddr_in serv_addr;
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) exit(EXIT_FAILURE);
   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) exit(EXIT_FAILURE);
   if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
exit(EXIT_FAILURE);
```

```
    read(sock, buffer, MAX_MESSAGE_SIZE);
    printf("Message from server: %s\n", buffer);
    send(sock, client_hello_message, strlen(client_hello_message), 0);
    printf("Hello message sent to server\n");

    return 0;
}
```

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

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

#define PORT 8080
#define MAX_MESSAGE_SIZE 100

int is_palindrome(const char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - i - 1]) {
            return 0;
        }
    }
    return 1;
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[MAX_MESSAGE_SIZE] = {0};
    const char *palindrome_msg = "The given string is a palindrome";
    const char *not_palindrome_msg = "The given string is not a palindrome";

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) 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)) == -1)
exit(EXIT_FAILURE);
    if (listen(server_fd, 3) == -1) exit(EXIT_FAILURE);
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t
*)&addrlen)) == -1) exit(EXIT_FAILURE);
    read(new_socket, buffer, MAX_MESSAGE_SIZE);
    printf("String received from client: %s\n", buffer);
    if (is_palindrome(buffer)) {
        send(new_socket, palindrome_msg, strlen(palindrome_msg), 0);
        printf("Palindrome message sent to client\n");
    } else {
        send(new_socket, not_palindrome_msg, strlen(not_palindrome_msg), 0);
        printf("Not palindrome message sent to client\n");
    }

    return 0;
}

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

#define PORT 8080
#define MAX_MESSAGE_SIZE 100

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[MAX_MESSAGE_SIZE] = {0};
    const char *input_string = "level";

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) exit(EXIT_FAILURE);
    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) exit(EXIT_FAILURE);
```

```
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
exit(EXIT_FAILURE);
    send(sock, input_string, strlen(input_string), 0);
    printf("String sent to server: %s\n", input_string);
    read(sock, buffer, MAX_MESSAGE_SIZE);
    printf("Message from server: %s\n", buffer);

    return 0;
}
```

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

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

#define PORT 8080
#define MAX_MESSAGE_SIZE 100

int main() {
    int sockfd;
    char buffer[MAX_MESSAGE_SIZE];
    struct sockaddr_in servaddr, cliaddr;

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

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information
    servaddr.sin_family = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
```

```
    // Bind the socket with the server address
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    int len, n;
    len = sizeof(cliaddr);  //len is value/resuslt

    n = recvfrom(sockfd, (char *)buffer, MAX_MESSAGE_SIZE, MSG_WAITALL, (struct
sockaddr *)&cliaddr, &len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);

    sendto(sockfd, (const char *)"Hello from server", strlen("Hello from server"),
MSG_CONFIRM, (const struct sockaddr *)&cliaddr, len);
    printf("Hello message sent.\n");

    return 0;
}

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

#define PORT 8080
#define MAX_MESSAGE_SIZE 100

int main() {
    int sockfd;
    char buffer[MAX_MESSAGE_SIZE];
    struct sockaddr_in servaddr;

    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
```

```
    }

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    int n, len;
    len = sizeof(servaddr);

    sendto(sockfd, (const char *)"Hello from client", strlen("Hello from client"),
MSG_CONFIRM, (const struct sockaddr *)&servaddr, len);
    printf("Hello message sent.\n");

    n = recvfrom(sockfd, (char *)buffer, MAX_MESSAGE_SIZE, MSG_WAITALL, (struct
sockaddr *)&servaddr, &len);
    buffer[n] = '\0';
    printf("Server : %s\n", buffer);

    close(sockfd);

    return 0;
}
```