1. Explain the connection procedure followed in client server communication

The connection procedure in client-server communication typically follows several steps to establish a reliable and secure connection between the client and the server. Here's a generalized outline of the process:

i. Client Initialization: - create socket and prepare all necessary parameters for the socket.

ii. Server Initialization: - server prepares for incoming connections from clients.Creates Socket and binds it to a port on server machine.

iii. Connection Request: - client sends request to server which has server's IP address , portname.

iv. Server Acceptance: - After receiving connection request, server accepts it and acknowledges client's request & communicate with it.

v. Handshaking: - both the client and server exchange information to establish parameters for communication. This may involve negotiating encryption methods, data formats, and other settings to ensure compatibility and security.

vi. Data Transfer: Once the connection is established and parameters are agreed upon, data transfer can begin. The client and server can now send messages, requests, or other data back and forth as needed.

vii. Connection Termination: Eventually, either the client or the server may decide to terminate the connection. This could be due to the completion of a task, an error, or simply because the session is over. Both the client and server typically exchange messages to gracefully close the connection.

viii. Resource Cleanup: After the connection is terminated, both the client and server may perform cleanup tasks. This could involve releasing allocated memory, closing file descriptors, or performing any other necessary cleanup operations.

2. What is the use of bind() function in socket programming ?
   In socket programming, the bind() function is used to associate a socket with a specific network address, such as an IP address and port number. This function is typically used by servers to specify the address they will listen on for incoming connections.

3. What is Datagram Socket ?
   A datagram socket is a type of network socket used in communication between computers on a network. It's a fundamental concept in network programming, particularly in the context of protocols like UDP (User Datagram Protocol).
4. Write a server/client model socket program to exchange hello message between them.

Server.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

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

#define PORT 4256
#define BUFFER_SIZE 1024

int main() {
   int server_socket, client_socket;
   struct sockaddr_in server_address, client_address;
   char buffer[BUFFER_SIZE] = {0};
   int addrlen = sizeof(server_address);

   // Create TCP socket
   if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
      perror("socket failed");
      exit(EXIT_FAILURE); }

   // Setup server address structure
   server_address.sin_family = AF_INET;
   server_address.sin_addr.s_addr = INADDR_ANY;
   server_address.sin_port = htons(PORT);

   // Bind socket to address and port
   if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
      perror("bind failed");
      exit(EXIT_FAILURE);}

   // Listen for incoming connections
   if (listen(server_socket, 5) < 0) {
      perror("listen failed");
      exit(EXIT_FAILURE); }

   printf("Server listening on port %d\n", PORT);

   // Accept incoming connection
   if ((client_socket = accept(server_socket, (struct sockaddr *)&client_address,
(socklen_t*)&addrlen)) < 0) {
      perror("accept failed");
      exit(EXIT_FAILURE); }

   printf("Connection accepted from %s:%d\n", inet_ntoa(client_address.sin_addr),
ntohs(client_address.sin_port));

   // Receive data from client
   read(client_socket, buffer, BUFFER_SIZE);
```

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

  // Send response to client
  char *hello = "Hello from server";
  send(client_socket, hello, strlen(hello), 0);
  printf("Hello message sent\n");

  // Close sockets
  close(client_socket);
  close(server_socket);

  return 0;}
```

Client.c

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

#define PORT 4256
#define BUFFER_SIZE 1024

int main() {
  int client_socket;
  struct sockaddr_in server_address;
  char buffer[BUFFER_SIZE] = {0};

  // Create TCP socket
  if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket failed");
    exit(EXIT_FAILURE); }

  // Setup server address structure
  server_address.sin_family = AF_INET;
  server_address.sin_port = htons(PORT);

  // Convert IPv4 and IPv6 addresses from text to binary form
  if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
    perror("invalid address");
    exit(EXIT_FAILURE);
  }

  // Connect to server
```

```
    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
      perror("connect failed");
      exit(EXIT_FAILURE); }

    // Send data to server
    char *hello = "Hello from client";
    send(client_socket, hello, strlen(hello), 0);
    printf("Hello message sent\n");

    // Receive response from server
    read(client_socket, buffer, BUFFER_SIZE);
    printf("Received: %s\n", buffer);

    // Close socket
    close(client_socket);

    return 0;
  }
```

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

Server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 12345
#define BUFFER_SIZE 1024

// Function to check if a string is a palindrome
int isPalindrome(char *str) {
  int length = strlen(str);
  for (int i = 0; i < length / 2; i++) {
    if (str[i] != str[length - i - 1]) {
      return 0; // Not a palindrome
    }}
```

```
    return 1; // Palindrome
}
int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_address, client_address;
    char buffer[BUFFER_SIZE] = {0};
    int addrlen = sizeof(server_address);
    // Create TCP socket
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE); }
    // Setup server address structure
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);
    // Bind socket to address and port
    if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);}
    // Listen for incoming connections
    if (listen(server_socket, 5) < 0) {
        perror("listen failed");
        exit(EXIT_FAILURE); }
    printf("Server listening on port %d\n", PORT);
    // Accept incoming connection
    if ((client_socket = accept(server_socket, (struct sockaddr *)&client_address,
(socklen_t*)&addrlen)) < 0) {
        perror("accept failed");
        exit(EXIT_FAILURE); }
    printf("Connection accepted from %s:%d\n", inet_ntoa(client_address.sin_addr),
ntohs(client_address.sin_port));
```

```
    // Receive data from client
    read(client_socket, buffer, BUFFER_SIZE);
    printf("Received: %s\n", buffer);
    // Check if received string is a palindrome
    int result = isPalindrome(buffer);
    // Send result to client
    if (result) {send(client_socket, "Palindrome", strlen("Palindrome"), 0); }
  else {send (client_socket, "Not Palindrome", strlen("Not Palindrome"), 0);}
    // Close sockets
    close(client_socket);
    close(server_socket);
    return 0;}
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 12345
#define BUFFER_SIZE 1024
int main() {
    int client_socket;
    struct sockaddr_in server_address;
    char buffer[BUFFER_SIZE] = {0};
    // Create TCP socket
    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket failed");
        exit(EXIT_FAILURE); }
    // Setup server address structure
    server_address.sin_family = AF_INET;
```

```
server_address.sin_port = htons(PORT);
// Convert IPv4 and IPv6 addresses from text to binary form
if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
   perror("invalid address");
   exit(EXIT_FAILURE); }
// Connect to server
if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
   perror("connect failed");
   exit(EXIT_FAILURE);}
// Input string from user
printf("Enter a string: ");
fgets(buffer, BUFFER_SIZE, stdin);
// Send string to server
send(client_socket, buffer, strlen(buffer), 0);
// Receive result from server
read(client_socket, buffer, BUFFER_SIZE);
printf("Result from server: %s\n", buffer);
// Close socket
close(client_socket);
return 0;}
```

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

Udp_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 12345
#define BUFFER_SIZE 1024
int main() {
   int server_socket;
```

```
  struct sockaddr_in server_address, client_address;
  char buffer[BUFFER_SIZE] = {0};
  int addrlen = sizeof(client_address);
  // Create UDP socket
  if ((server_socket = socket(AF_INET, SOCK_DGRAM, 0)) == 0) {
     perror("socket failed");
     exit(EXIT_FAILURE); }
 // Setup server address structure
  server_address.sin_family = AF_INET;
  server_address.sin_addr.s_addr = INADDR_ANY;
  server_address.sin_port = htons(PORT);
  // Bind socket to address and port
  if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
     perror("bind failed");
     exit(EXIT_FAILURE);}
  printf("Server listening on port %d\n", PORT);
  while (1) {
     // Receive data from client
     int bytes_received = recvfrom(server_socket, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&client_address, (socklen_t*)&addrlen);
     if (bytes_received < 0) {
        perror("recvfrom failed");
        exit(EXIT_FAILURE);}
     printf("Received from client: %s\n", buffer);
     // Send acknowledgment to client
     char *ack = "Message received";
     sendto(server_socket, ack, strlen(ack), 0, (struct sockaddr *)&client_address, addrlen);}
  // Close socket
  close(server_socket);
  return 0; }
```

Udp_client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 12345
#define BUFFER_SIZE 1024
int main() {
   int client_socket;
   struct sockaddr_in server_address;
   char buffer[BUFFER_SIZE] = {0};
   // Create UDP socket
   if ((client_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      perror("socket failed");
      exit(EXIT_FAILURE);}
   // Setup server address structure
   server_address.sin_family = AF_INET;
   server_address.sin_port = htons(PORT);
   // Convert IPv4 and IPv6 addresses from text to binary form
   if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
      perror("invalid address");
      exit(EXIT_FAILURE);  }
   // Input message from user
   printf("Enter a message: ");
   fgets(buffer, BUFFER_SIZE, stdin);
   // Send message to server
   sendto(client_socket, buffer, strlen(buffer), 0, (const struct sockaddr *)&server_address,
sizeof(server_address));
   // Receive acknowledgment from server
   int bytes_received = recvfrom(client_socket, buffer, BUFFER_SIZE, 0, NULL, NULL);
```

```
if (bytes_received < 0) {
    perror("recvfrom failed");
    exit(EXIT_FAILURE); }

printf("Acknowledgment from server: %s\n", buffer);
// Close socket
close(client_socket);
return 0;}
```