University of Lincoln
School of Computer Science
CMP9133M – Advanced Programming
Workshop 11

Task 1: UDP Network Communication

Design a simple UDP network communication program that consists of a server and a client.

1. **Server**
    - Create a UDP socket.
    - Bind the socket to a specific port on the server (e.g., port 12345).
    - Receive a UDP message from the client.
    - Print the received message.
2. **Client**
    - Create a UDP socket.
    - Send a UDP message to the server.
    - The message can be input through the command line or from a user prompt.
    - Close the socket.

Instructions:

1. Implement the server code with the following steps:
    - Create a UDP socket using the socket() function from the <sys/socket.h> library.
    - Bind the socket to a specific port using the bind() function.
    - Use the recvfrom() function to receive the message from the client.
    - Print the received message to the console.
    - Close the socket.
2. Implement the client code with the following steps:
    - Create a UDP socket using the socket() function.
    - Prepare the destination server address and port information.
    - Get the message from the user, either through command line input or user prompt.
    - Use the sendto() function to send the message to the server.
    - Close the socket.

Example Interaction:

Server:

```
Server started. Waiting for messages...
Received message from client: Hello, server!
```

Client:

```
Enter the message to send: Hello, server!
Message sent to server.
```

Task 2: TCP Network Communication

Design a simple TCP network communication program that consists of a server and a client.

1. **Server**
    - Create a TCP socket.
    - Bind the socket to a specific port on the server (e.g., port 12345).
    - Listen for incoming connections.
    - Accept a client connection.
    - Receive a TCP message from the client.
    - Print the received message.
    - Send a response message to the client.
    - Close the socket.
2. **Client**
    - Create a TCP socket.
    - Connect to the server using an IP address and port.
    - Send a TCP message to the server.
    - Receive a response message from the server.
    - Print the response message.
    - Close the socket.

Instructions:

1. Implement the server code with the following steps:
    - Create a TCP socket using the socket() function from the <sys/socket.h> library.
    - Bind the socket to a specific port using the bind() function.
    - Use the listen() function to listen for incoming connections.
    - Use the accept() function to accept a client connection and obtain a new socket descriptor.
    - Use the recv() function to receive the message from the client.
    - Print the received message to the console.
    - Use the send() function to send a response message to the client.

- Close the sockets.

2. Implement the client code with the following steps:
    - Create a TCP socket using the socket() function.
    - Use the connect() function to establish a connection to the server using the server's IP address and port.
    - Use the send() function to send a message to the server.
    - Use the recv() function to receive a response message from the server.
    - Print the response message to the console.
    - Close the socket.

Example Interaction:

Server:

```
Server started. Waiting for connections...
Received message from client: Hello, server!
Response sent to client: Message received successfully.
```

Client:

```
Connected to server.
Enter the message to send: Hello, server!
Response received from server: Message received successfully.
```

Task 3: Simple Chat Program

Write a simple C++ program to implement a basic chat application using client-server architecture.

1. **Server**
    - Create a TCP server that listens for client connections on a specified port.
    - Accept incoming client connections.
    - Receive messages from connected clients and broadcast them to all other connected clients.
    - Handle disconnections gracefully.
2. **Client**
    - Create a TCP client that connects to the server using the server's IP address and port.

- Send messages to the server.
- Receive messages from the server and display them on the client console.
- Allow the user to gracefully disconnect from the server.

Instructions:

1. Implement the server code with the following steps:
    - Create a TCP socket for the server using the socket() function.
    - Bind the server socket to a specific port using the bind() function.
    - Listen for incoming connections using the listen() function.
    - Accept incoming client connections using the accept() function.
    - Receive messages from the connected clients and broadcast them to all other connected clients.
    - Handle client disconnections.
2. Implement the client code with the following steps:
    - Create a TCP socket for the client using the socket() function.
    - Establish a connection to the server using the server's IP address and port using the connect() function.
    - Send messages to the server using the send() function.
    - Receive messages from the server using the recv() function.
    - Display received messages on the client console.
    - Allow the user to gracefully disconnect from the server.

Example Interaction:

Server:

```
Server started. Waiting for connections...
Client connected: 192.168.0.2
Client connected: 192.168.0.3
Received message from 192.168.0.2: Hello, everyone!
Broadcasting message from 192.168.0.2 to all clients...
Message broadcasted successfully!
Received message from 192.168.0.3: Hi there!
Broadcasting message from 192.168.0.3 to all clients...
Message broadcasted successfully!
```

Client1:

```
Connected to server.
Enter the message to send: Hello, everyone!
Message sent successfully!
Response from server: Message broadcasted successfully!
Enter the message to send: Goodbye!
Message sent successfully!
Response from server: Message broadcasted successfully!
Disconnected from server.
```

Client2:

```
Connected to server.
Enter the message to send: Hi there!
Message sent successfully!
Response from server: Message broadcasted successfully!
Enter the message to send: Bye!
Message sent successfully!
Response from server: Message broadcasted successfully!
Disconnected from server.
```