CS558: Computer Systems Lab
Assignment – 4a

Report

Pradeep Kumar Soni – 244101032
Rahul – 244101035
Rohan Dayal – 244101039

Application 3

Problem Statement

The task is to implement two C programs: a server and a client. The server must be capable of handling multiple clients concurrently and should support communication over both TCP and UDP sockets. The client will connect to the server using either TCP or UDP, based on user input, and will transmit Base64-encoded messages.
The primary objective is to create a simple communication protocol using Base64 encoding that supports both TCP (connection-oriented) and UDP (connectionless) communication methods. Additionally, the server must support concurrent communication with multiple clients using multithreading.

Socket Programming -

Socket programming allows applications to establish connections and exchange data across diverse network environments. By utilizing sockets, developers can create robust and scalable networked applications that can operate over local area networks (LANs) or the global Internet.

Connection Types -

1. TCP (Transmission Control Protocol):
    a. Provides reliable, ordered, and error-checked delivery of data streams.
    b. Establishes a persistent connection between client and server.
    c. Ensures data integrity through acknowledgments and retransmissions.
2. UDP (User Datagram Protocol):
    a. Offers a lightweight, connectionless communication method.
    b. Prioritizes speed over reliability, making it suitable for real-time applications.
    c. Does not guarantee ordered delivery or data integrity.

Base64 Encoding -

Base64 encoding is a binary-to-text encoding scheme that converts binary data into a sequence of printable ASCII characters.

The Encoding Algorithm

1) Input Processing:
    a) It divides the input into groups of 3 bytes (24 bits).
2) Bit Conversion:
    a) Each 3-byte (24-bit) group is split into four 6-bit segments.
3) Character Mapping:
    a) Each 6-bit segment (with values from 0-63) is mapped to a corresponding character in the Base64 alphabet.
    b) The standard Base64 alphabet consists of:
        i) 26 uppercase letters (A-Z)
        ii) 26 lowercase letters (a-z)
        iii) 10 digits (0-9)
        iv) 2 special characters (+ and /)
4) Padding:
    a) If the input length is not divisible by 3, padding is applied.
    b) The padding character is "=".
    c) One "=" is added if there's a remainder of 2 bytes (creating 3 Base64 characters plus 1 padding character).
    d) Two "=" characters are added if there's a remainder of 1 byte (creating 2 Base64 characters plus 2 padding characters).

Size Considerations

Base64 encoding increases the data size by approximately 33%. This is because 3 bytes (24 bits) of binary data become 4 bytes (32 bits) of Base64-encoded text.

Message Structure and Type -

Each message exchanged between the client and server consists of the following fields:

Field	Description
Message Type	Integer indicating the type of message (1, 2, or 3)
Message Content	Character array of fixed size (e.g., MSG LEN)

There are three distinct types of messages that can be sent between client and server:

Message Type	Description
1	Base64-encoded message sent from client to server
2	Acknowledgment (ACK) message sent from server to client
3	Termination message to close the connection

Implementation Details -

Server -

The server component is designed to handle concurrent client connections while supporting both TCP and UDP protocols on the same port.

Socket Initialization and Configuration

The server begins by initializing two distinct socket types: a TCP socket for connection-oriented communication and a UDP socket for connectionless exchange. Both sockets are bound to the same port number, which is specified as a command-line argument when starting the server.

The server employs the `INADDR_ANY` address to accept connections from any network interface, maximizing accessibility. Socket options are configured to allow address reuse, preventing issues with socket binding after server restarts.

Concurrent Client Handling

The ability to handle multiple TCP clients concurrently is achieved through a multithreaded architecture using threads (pthreads). When a new TCP client connects, the server:

1. Accepts the incoming connection
2. Creates a new thread dedicated to handling that specific client.

Each client-handling thread operates independently, processing messages from its assigned client without blocking other connections.

For UDP communication, which is connectionless by nature, the server processes incoming datagrams in the main thread itself, since UDP does not maintain connection state.

Base64 Decoding Process

When the server receives a Base64-encoded message (Type 1) from a client, it performs the following steps:

1. Extracts the encoded content from the message structure
2. Decodes the encoded text.
3. Adds a null terminator to create a proper C string
4. Displays the decoded message on the server console
5. Sends an acknowledgment (Type 2) back to the client
6. Frees all allocated memory to prevent leaks

Efficient I/O Multiplexing

To monitor activity on both TCP and UDP sockets simultaneously, the server implements I/O multiplexing using the `select()` system call. This approach allows the server to:

1. Wait efficiently for activity on either socket without consuming CPU resources
2. Process TCP connections and UDP datagrams as they arrive
3. Maintain responsiveness even under varying load conditions

The server's main loop continuously monitors both sockets, handling new TCP connections by spawning threads and processing UDP messages directly in the main thread.

Graceful Termination Handling

The server recognizes Type 3 messages as termination requests. For TCP clients, this results in closing the connection and terminating the associated thread. For UDP clients, the server acknowledges the termination request but maintains the socket since UDP is connectionless.

Client -

Protocol Selection and Connection Establishment

The client determines which protocol to use based on command-line arguments provided by the user.

- TCP for reliable, ordered communication
- UDP for faster, connectionless communication

For TCP connections, the client establishes a persistent connection with the server before exchanging messages. For UDP, no connection establishment is necessary, and the client simply sends datagrams to the server's address.

User Interface and Input Processing

1. Prompts users to enter messages
2. Processes special commands (e.g., "terminate" to end the session)
3. Displays acknowledgments received from the server
4. Provides feedback on connection status and errors

Base64 Encoding Implementation

Before transmitting user input to the server, the client encodes it using Base64. This encoding process:

1. Takes the raw user input as a byte array
2. Encode it according to the Base64 encoding algorithm.
3. Places the encoded string in the message structure
4. Sets the message type to 1 (Base64-encoded message)

Protocol-Specific Communication Handling

The client implements different communication patterns depending on the selected protocol:

For TCP:

- Uses `send()` to transmit messages through the established connection
- Uses `recv()` to receive acknowledgments, which blocks until data arrives or the connection closes
- Maintains connection state throughout the session

For UDP:

- Uses `sendto()` to transmit messages to the server's address
- Uses `recvfrom()` with a timeout mechanism to receive acknowledgments
- Maintains no connection state between messages

Graceful Termination

The client provides a mechanism for gracefully terminating the communication session. When the user enters "quit":

1. The client creates a Type 3 (termination) message
2. Sends this message to the server using the appropriate protocol function
3. Closes its socket and exits

This ensures that both client and server can clean up resources properly when the session ends.

Server Functions:

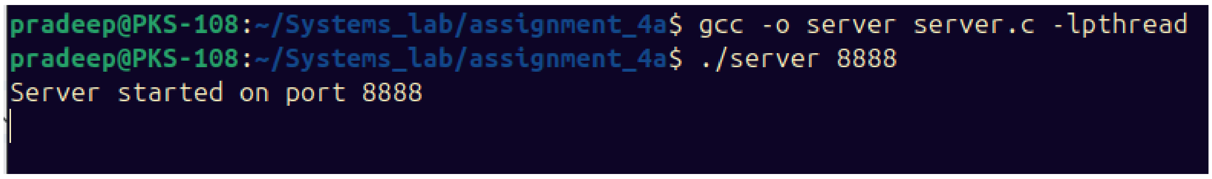
- `main()`: Creates TCP and UDP sockets, binds them to the specified port, uses select() to monitor activity on both sockets, and handles incoming connections and messages.
- `handle_tcp_client()`: Dedicated thread function that receives messages from TCP clients, decodes Base64 content, sends acknowledgments, and handles termination requests.
- `handle_udp_message()`: Processes UDP datagrams by receiving messages, decoding Base64 content, and sending acknowledgments back to clients.
- `decode_base64()`: Converts Base64-encoded strings back to their original form by mapping each character to its corresponding value, combining sextets, and extracting the original bytes.
- `print_message_type()`: Displays the type of message received (Type 1: Base64-encoded, Type 2: ACK, or Type 3: Termination) for logging purposes.

Client Functions:

- `main()`: Creates either TCP or UDP socket based on user input, connects to the server, processes user messages, and manages the communication session.
- `encode_base64()`: Converts input strings to Base64 format by grouping bytes into 24-bit chunks, splitting them into 6-bit values, and mapping to Base64 characters.
- `send_message()`: Packages messages with appropriate type information and sends them to the server using either TCP or UDP protocols.
- `receive_ack()`: Waits for and processes acknowledgement messages from the server using either recv() for TCP or recvfrom() for UDP.
- `print_message_type()`: Displays the type of message being sent or received to enhance user feedback during communication.

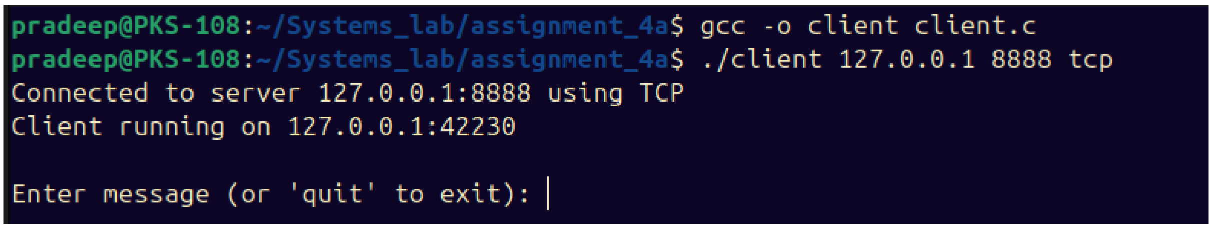
Compilation and Execution Instructions:

- Compile server: `gcc -o server server.c -lpthread`
- Run server: `./server <port>` For example : `./server 8888`



```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ gcc -o server server.c -lpthread
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./server 8888
Server started on port 8888
```

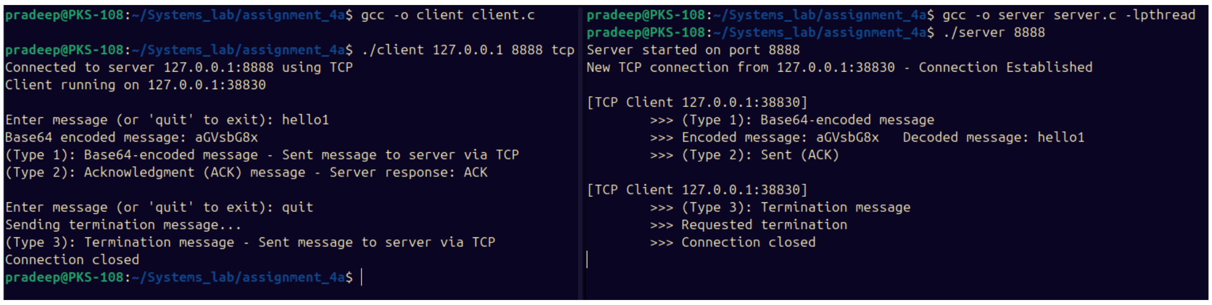
- Compile client: `gcc -o client client.c`
- Run client: `./client <server_ip> <port> <protocol>`



```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ gcc -o client client.c
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 tcp
Connected to server 127.0.0.1:8888 using TCP
Client running on 127.0.0.1:42230

Enter message (or 'quit' to exit):
```

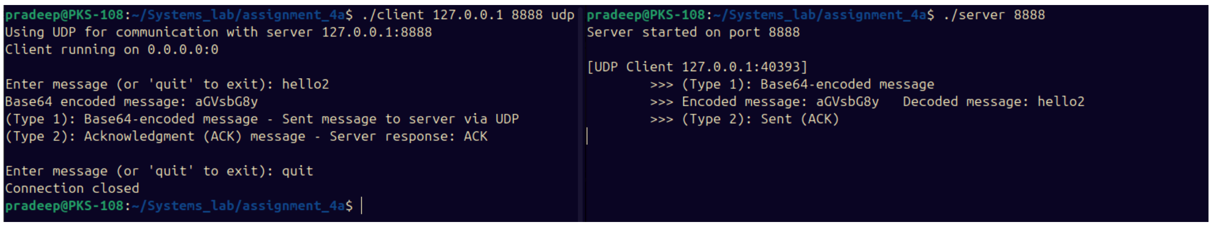
Outputs -

Single client communication -



```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ gcc -o client client.c

pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 tcp
Connected to server 127.0.0.1:8888 using TCP
Client running on 127.0.0.1:38830

Enter message (or 'quit' to exit): hello1
Base64 encoded message: aGVsbG8x
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Sending termination message...
(Type 3): Termination message - Sent message to server via TCP
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$
```

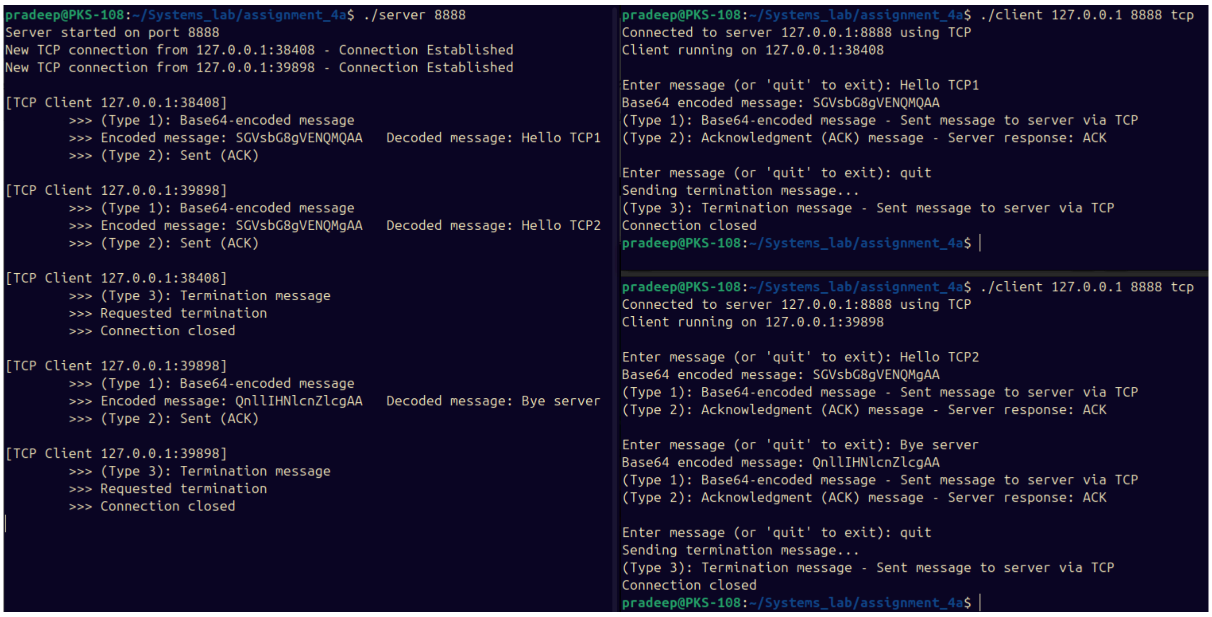
```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ gcc -o server server.c -lpthread
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./server 8888
Server started on port 8888
New TCP connection from 127.0.0.1:38830 - Connection Established

[TCP Client 127.0.0.1:38830]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: aGVsbG8x   Decoded message: hello1
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:38830]
        >>> (Type 3): Termination message
        >>> Requested termination
        >>> Connection closed
```



```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 udp
Using UDP for communication with server 127.0.0.1:8888
Client running on 0.0.0.0:0

Enter message (or 'quit' to exit): hello2
Base64 encoded message: aGVsbG8y
(Type 1): Base64-encoded message - Sent message to server via UDP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$
```

```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./server 8888
Server started on port 8888

[UDP Client 127.0.0.1:40393]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: aGVsbG8y   Decoded message: hello2
        >>> (Type 2): Sent (ACK)
```

Multiple concurrent TCP clients -

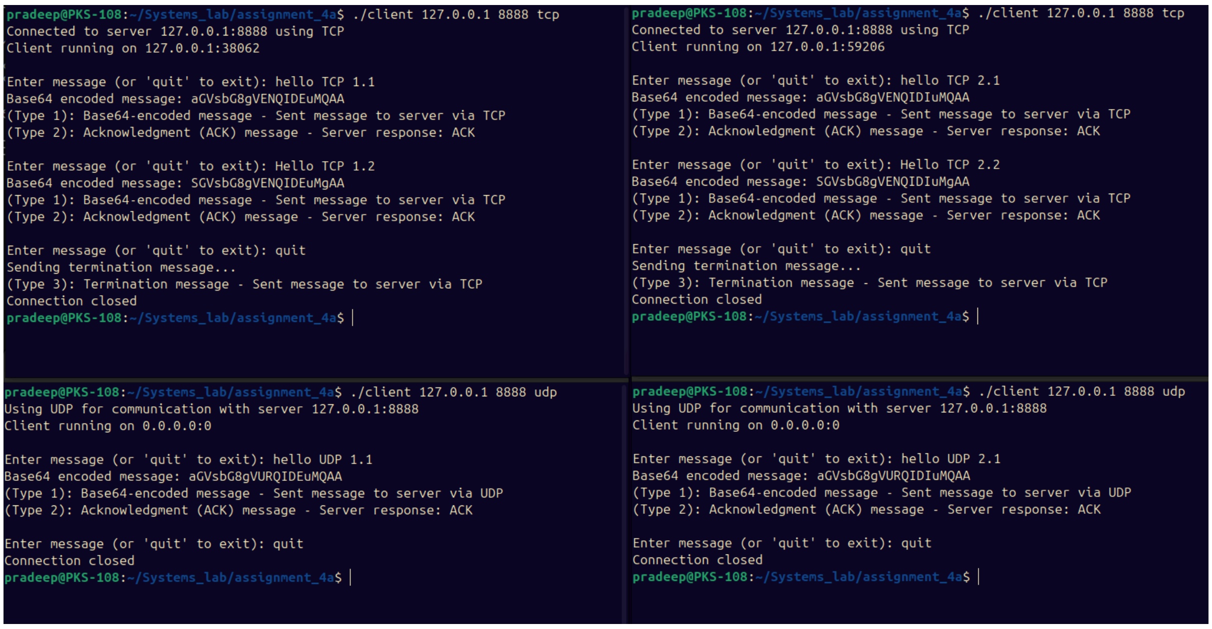


```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./server 8888
Server started on port 8888
New TCP connection from 127.0.0.1:38408 - Connection Established
New TCP connection from 127.0.0.1:39898 - Connection Established

[TCP Client 127.0.0.1:38408]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: SGVsbG8gVENQMQAA   Decoded message: Hello TCP1
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:39898]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: SGVsbG8gVENQMgAA   Decoded message: Hello TCP2
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:38408]
        >>> (Type 3): Termination message
        >>> Requested termination
        >>> Connection closed

[TCP Client 127.0.0.1:39898]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: QnllIHNlcnZlcgAA   Decoded message: Bye server
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:39898]
        >>> (Type 3): Termination message
        >>> Requested termination
        >>> Connection closed
```

```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 tcp
Connected to server 127.0.0.1:8888 using TCP
Client running on 127.0.0.1:38408

Enter message (or 'quit' to exit): Hello TCP1
Base64 encoded message: SGVsbG8gVENQMQAA
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Sending termination message...
(Type 3): Termination message - Sent message to server via TCP
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$
```

```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 tcp
Connected to server 127.0.0.1:8888 using TCP
Client running on 127.0.0.1:39898

Enter message (or 'quit' to exit): Hello TCP2
Base64 encoded message: SGVsbG8gVENQMgAA
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): Bye server
Base64 encoded message: QnllIHNlcnZlcgAA
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Sending termination message...
(Type 3): Termination message - Sent message to server via TCP
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$
```

Mixed TCP and UDP clients -

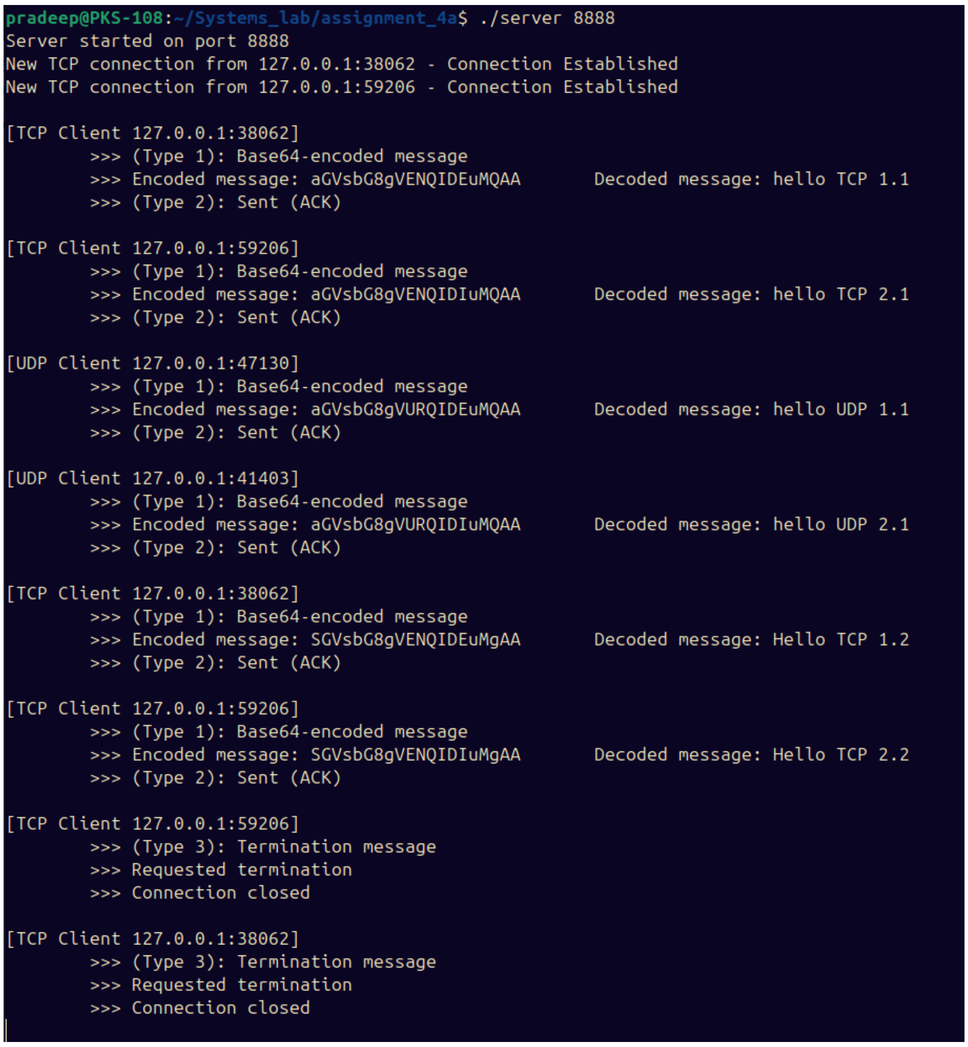


```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 tcp
Connected to server 127.0.0.1:8888 using TCP
Client running on 127.0.0.1:38062

Enter message (or 'quit' to exit): hello TCP 1.1
Base64 encoded message: aGVsbG8gVENQIDEuMQAA
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): Hello TCP 1.2
Base64 encoded message: SGVsbG8gVENQIDEuMgAA
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Sending termination message...
(Type 3): Termination message - Sent message to server via TCP
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$ |
```

```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 tcp
Connected to server 127.0.0.1:8888 using TCP
Client running on 127.0.0.1:59206

Enter message (or 'quit' to exit): hello TCP 2.1
Base64 encoded message: aGVsbG8gVENQIDIuMQAA
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): Hello TCP 2.2
Base64 encoded message: SGVsbG8gVENQIDIuMgAA
(Type 1): Base64-encoded message - Sent message to server via TCP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Sending termination message...
(Type 3): Termination message - Sent message to server via TCP
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$ |
```

```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 udp
Using UDP for communication with server 127.0.0.1:8888
Client running on 0.0.0.0:0

Enter message (or 'quit' to exit): hello UDP 1.1
Base64 encoded message: aGVsbG8gVURQIDEuMQAA
(Type 1): Base64-encoded message - Sent message to server via UDP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$ |
```

```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./client 127.0.0.1 8888 udp
Using UDP for communication with server 127.0.0.1:8888
Client running on 0.0.0.0:0

Enter message (or 'quit' to exit): hello UDP 2.1
Base64 encoded message: aGVsbG8gVURQIDIuMQAA
(Type 1): Base64-encoded message - Sent message to server via UDP
(Type 2): Acknowledgment (ACK) message - Server response: ACK

Enter message (or 'quit' to exit): quit
Connection closed
pradeep@PKS-108:~/Systems_lab/assignment_4a$ |
```



```
pradeep@PKS-108:~/Systems_lab/assignment_4a$ ./server 8888
Server started on port 8888
New TCP connection from 127.0.0.1:38062 - Connection Established
New TCP connection from 127.0.0.1:59206 - Connection Established

[TCP Client 127.0.0.1:38062]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: aGVsbG8gVENQIDEuMQAA       Decoded message: hello TCP 1.1
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:59206]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: aGVsbG8gVENQIDIuMQAA       Decoded message: hello TCP 2.1
        >>> (Type 2): Sent (ACK)

[UDP Client 127.0.0.1:47130]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: aGVsbG8gVURQIDEuMQAA       Decoded message: hello UDP 1.1
        >>> (Type 2): Sent (ACK)

[UDP Client 127.0.0.1:41403]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: aGVsbG8gVURQIDIuMQAA       Decoded message: hello UDP 2.1
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:38062]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: SGVsbG8gVENQIDEuMgAA       Decoded message: Hello TCP 1.2
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:59206]
        >>> (Type 1): Base64-encoded message
        >>> Encoded message: SGVsbG8gVENQIDIuMgAA       Decoded message: Hello TCP 2.2
        >>> (Type 2): Sent (ACK)

[TCP Client 127.0.0.1:59206]
        >>> (Type 3): Termination message
        >>> Requested termination
        >>> Connection closed

[TCP Client 127.0.0.1:38062]
        >>> (Type 3): Termination message
        >>> Requested termination
        >>> Connection closed
```

Conclusion -

The client code implements Base64 encoding and communication via TCP/UDP, while the server code handles multiple clients through threading and decodes Base64 messages. Both use a message structure with types (1=encoded message, 2=ACK, 3=termination).

Compile with gcc and run the server first (./server <port>), then clients (./client <server_ip> <port> tcp/udp). The system demonstrates socket programming with protocol flexibility and encoding/decoding capabilities.