```python
import socket

server = socket.socket()
server.bind(("127.0.0.1", 5000))
server.listen(1)
print("Listening at 5000")

while True:
    con, add = server.accept()
    print(f"Connected with {add}")
    fname = con.recv(1024).decode()

    try:
        with open(fname, 'r') as file:
            con.send(file.read().encode())
    except FileNotFoundError:
        print("File not found")
    con.close()
```

client.py

```python
import socket

client = socket.socket()
client.connect(('127.0.0.1', 65432))

file_name = input("Enter file name: ")
client.send(file_name.encode())

response = client.recv(1024).decode()
print("Server response:", response)

client.close()
```

2 nd Lab:

Client

```
import socket

# Create a UDP client socket
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Define the server address
server_address = ('127.0.0.1', 65432)

try:
    # Get user input and send to server
    message = input("Enter message: ")
    client.sendto(message.encode(), server_address)  # Ensure
server address is supplied

    # Receive and display the response
    data, _ = client.recvfrom(1024)
    print(f"Server response: {data.decode()}")

finally:
    client.close()  # Close the socket properly
```

Server

```
import socket

server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(('127.0.0.1', 65432))

while True:
    data, address = server.recvfrom(1024)
```

```
    if data:
        print(f"Received from {address}: {data.decode()}")
        response = data.decode().upper()  # Decode the data
before processing
        server.sendto(response.encode(), address)  # Send back
the uppercase data
```

3 rs

Client

```
import socket

def crc_ccitt(data: bytes, poly=0x1021, init_crc=0xFFFF):
    crc = init_crc
    for byte in data:
        crc ^= (byte << 8)
        for _ in range(8):
            crc = (crc << 1) ^ poly if crc & 0x8000 else crc <<
1
            crc &= 0xFFFF  # Ensure CRC stays within 16 bits
    return crc

def client():
    host, port = "127.0.0.1", 65432
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect((host, port))
    print("Connected to server.")

    # Receive message and CRC
```

```
    data = client_socket.recv(1024)
    message, received_crc = data[:-2], int.from_bytes(data[-
2:], "big")
    print(f"Received message: {message.decode()}")
    print(f"Received CRC: {received_crc:#04x}")

    # Validate CRC
    calculated_crc = crc_ccitt(message)
    if calculated_crc == received_crc:
        print("CRC check passed. Data integrity verified.")
    else:
        print("CRC check failed. Data is corrupted.")

    client_socket.close()

if __name__ == "__main__":
    client()
```

Server

```
import socket

def crc_ccitt(data: bytes, poly=0x1021, init_crc=0xFFFF):
    crc = init_crc
    for byte in data:
        crc ^= (byte << 8)
        for _ in range(8):
            crc = (crc << 1) ^ poly if crc & 0x8000 else crc <<
1
            crc &= 0xFFFF  # Ensure CRC stays within 16 bits
    return crc

def server():
    host, port = "127.0.0.1", 65432
```

```
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}...")

    conn, addr = server_socket.accept()
    print(f"Connected by {addr}")

    # Get user input for the message
    message = input("Enter a message to send: ").encode()
    crc = crc_ccitt(message)
    conn.send(message + crc.to_bytes(2, "big"))
    print(f"Sent: {message} with CRC: {crc:#04x}")

    conn.close()
    server_socket.close()

if __name__ == "__main__":
    server()
```

4 th Program

```
import socket

def crc_ccitt(data: bytes, poly=0x1021, init_crc=0xFFFF):
    crc = init_crc
    for byte in data:
        crc ^= (byte << 8)
        for _ in range(8):
            crc = (crc << 1) ^ poly if crc & 0x8000 else crc <<
1
            crc &= 0xFFFF  # Ensure CRC stays within 16 bits
    return crc
```

```
def client():
    host, port = "127.0.0.1", 65432
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect((host, port))
    print("Connected to server.")

    # Receive message and CRC
    data = client_socket.recv(1024)
    message, received_crc = data[:-2], int.from_bytes(data[-
2:], "big")
    print(f"Received message: {message.decode()}")
    print(f"Received CRC: {received_crc:#04x}")

    # Validate CRC
    calculated_crc = crc_ccitt(message)
    if calculated_crc == received_crc:
        print("CRC check passed. Data integrity verified.")
    else:
        print("CRC check failed. Data is corrupted.")

    client_socket.close()

if __name__ == "__main__":
    client()
```

server

```
import socket

def crc_ccitt(data: bytes, poly=0x1021, init_crc=0xFFFF):
    crc = init_crc
    for byte in data:
        crc ^= (byte << 8)
        for _ in range(8):
```

```
            crc = (crc << 1) ^ poly if crc & 0x8000 else crc <<
1
            crc &= 0xFFFF  # Ensure CRC stays within 16 bits
    return crc

def server():
    host, port = "127.0.0.1", 65432
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}...")

    conn, addr = server_socket.accept()
    print(f"Connected by {addr}")

    # Get user input for the message
    message = input("Enter a message to send: ").encode()
    crc = crc_ccitt(message)
    conn.send(message + crc.to_bytes(2, "big"))
    print(f"Sent: {message} with CRC: {crc:#04x}")

    conn.close()
    server_socket.close()

if __name__ == "__main__":
    server()
```

%th

Client

Server

6th

Client

```
import socket
import struct

def calculate_checksum(data):
    checksum = 0
    n = len(data)
    for i in range(0, n - 1, 2):
        word = (data[i] << 8) + data[i + 1]
        checksum += word
        checksum = (checksum & 0xFFFF) + (checksum >> 16)
    if n % 2:
        checksum += data[-1] << 8
        checksum = (checksum & 0xFFFF) + (checksum >> 16)
    return ~checksum & 0xFFFF

def client():
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    target = "127.0.0.1"  # Replace with server IP if needed
    message = input("Enter the data to send: ").encode()

    # Calculate checksum
    checksum = calculate_checksum(message)

    # Print calculated checksum in binary
    print(f"Calculated checksum (binary): {bin(checksum)}")

    packet = message + struct.pack("!H", checksum)

    print(f"Sending message to {target}: {message.decode()}
with checksum (binary): {bin(checksum)}")
    client_socket.sendto(packet, (target, 9999))
```

```
    # Wait for a reply
    client_socket.settimeout(5)
    try:
        reply_packet, addr = client_socket.recvfrom(1024)

        # Extract message and checksum from the reply
        reply_message = reply_packet[:-2]
        reply_checksum = struct.unpack("!H", reply_packet[-
2:])[0]

        # Validate the checksum of the reply
        calculated_checksum = calculate_checksum(reply_message)

        # Print reply checksum in binary
        print(f"Received checksum (binary):
{bin(reply_checksum)}")
        print(f"Calculated checksum (binary):
{bin(calculated_checksum)}")

        if reply_checksum != calculated_checksum:
            print("Checksum mismatch in reply. Packet may be
corrupted.")
        else:
            print(f"Received valid reply from {addr}:
{reply_message.decode()} with checksum (binary):
{bin(reply_checksum)}")
    except socket.timeout:
        print("Request timed out.")
    finally:
        client_socket.close()

if __name__ == "__main__":
    client()
```

SERVER

```
import socket
import struct

def calculate_checksum(data):
    checksum = 0
    n = len(data)
    for i in range(0, n - 1, 2):
        word = (data[i] << 8) + data[i + 1]
        checksum += word
        checksum = (checksum & 0xFFFF) + (checksum >> 16)
    if n % 2:
        checksum += data[-1] << 8
        checksum = (checksum & 0xFFFF) + (checksum >> 16)
    return ~checksum & 0xFFFF

def server():
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    server_socket.bind(("0.0.0.0", 9999))
    print("UDP Server listening on port 9999...")

    while True:
        packet, addr = server_socket.recvfrom(1024)

        # Extract message and checksum
        received_message = packet[:-2]
        received_checksum = struct.unpack("!H", packet[-2:])[0]

        # Print received checksum in binary
        print(f"Received checksum (binary):
{bin(received_checksum)}")

        # Calculate checksum on the received message
```

```
        calculated_checksum =
calculate_checksum(received_message)

        # Print calculated checksum in binary
        print(f"Calculated checksum (binary):
{bin(calculated_checksum)}")

        if received_checksum != calculated_checksum:
            print(f"Checksum mismatch from {addr}. Packet
discarded.")
            continue

        print(f"Received valid message from {addr}:
{received_message.decode()}")

        # Send back the same data with a checksum
        reply_message = received_message
        reply_checksum = calculate_checksum(reply_message)

        # Print reply checksum in binary
        print(f"Reply checksum (binary):
{bin(reply_checksum)}")

        reply_packet = reply_message + struct.pack("!H",
reply_checksum)
        server_socket.sendto(reply_packet, addr)

if __name__ == "__main__":
    server()
```