File-Based Echo Server Implementation

Scenario Overview

1. **Client**: Sends message → Server saves to file → Server reads from file → Server echoes back
2. **Server**: Receives message → Writes to file → Reads from file → Sends back to client

Server Side Implementation

Option 1: Save Then Read Approach

python

```python
import socket
import os

# Configuration
port = 1236
address = "127.0.0.1"
BUF_SIZE = 15
HEADER_SIZE = 10

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((address, port))
server.listen(5)
print(f"Server listening on {address}:{port}...")

while True:
    con, addr = server.accept()
    print(f"Connected to client: {addr}")

    while True:
        # RECEIVE MESSAGE AND SAVE TO FILE
        message = ""
        msg_length = 0
        newmsg = True

        # Save received data to file as it arrives
        with open("server_received.txt", "w", encoding="utf-8") as f:
            while True:
                data = con.recv(BUF_SIZE)
                if not data:
                    break

                if newmsg:
                    msg_length = int(data[:HEADER_SIZE].decode("utf-8"))
                    chunk = data[HEADER_SIZE:].decode("utf-8")
                    newmsg = False
                else:
                    chunk = data.decode("utf-8")

                message += chunk
                f.write(chunk)  # Write chunk to file immediately

                if len(message) >= msg_length:
                    break

        print(f"Received and saved: {message}")
```

```
        # READ FROM FILE AND ECHO BACK
        try:
            with open("server_received.txt", "r", encoding="utf-8") as f:
                file_content = f.read()

            print(f"Read from file: {file_content}")

            # Send file content back to client
            header = f"{len(file_content):0{HEADER_SIZE}d}".encode("utf-8")
            con.send(header + file_content.encode("utf-8"))

        except FileNotFoundError:
            error_msg = "File not found"
            header = f"{len(error_msg):0{HEADER_SIZE}d}".encode("utf-8")
            con.send(header + error_msg.encode("utf-8"))

        if message == "exit":
            break

    con.close()
```

Option 2: Stream Processing Approach

python

```python
import socket
import tempfile
import os

# Configuration
port = 1236
address = "127.0.0.1"
BUF_SIZE = 15
HEADER_SIZE = 10

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((address, port))
server.listen(5)
print(f"Server listening on {address}:{port}...")

while True:
    con, addr = server.accept()
    print(f"Connected to client: {addr}")

    while True:
        # RECEIVE AND PROCESS MESSAGE
        message = ""
        msg_length = 0
        newmsg = True

        # Use temporary file for processing
        with tempfile.NamedTemporaryFile(mode='w+', encoding='utf-8', delete=False) as temp_file:
            temp_filename = temp_file.name

            # Receive and write to temp file
            while True:
                data = con.recv(BUF_SIZE)
                if not data:
                    break

                if newmsg:
                    msg_length = int(data[:HEADER_SIZE].decode("utf-8"))
                    chunk = data[HEADER_SIZE:].decode("utf-8")
                    newmsg = False
                else:
                    chunk = data.decode("utf-8")

                message += chunk
                temp_file.write(chunk)

                if len(message) >= msg_length:
```

```
            break

    print(f"Received: {message}")

    # READ FROM TEMP FILE AND SEND BACK
    try:
        with open(temp_filename, "r", encoding="utf-8") as f:
            file_content = f.read()

        print(f"Echoing from file: {file_content}")

        # Send file content back
        header = f"{len(file_content):0{HEADER_SIZE}d}".encode("utf-8")
        con.send(header + file_content.encode("utf-8"))

    except Exception as e:
        error_msg = f"Error reading file: {str(e)}"
        header = f"{len(error_msg):0{HEADER_SIZE}d}".encode("utf-8")
        con.send(header + error_msg.encode("utf-8"))

    finally:
        # Clean up temp file
        if os.path.exists(temp_filename):
            os.remove(temp_filename)

    if message == "exit":
        break

con.close()
```

Client Side Implementation

python

```python
import socket

# Configuration
port = 1236
address = "127.0.0.1"
BUF_SIZE = 15
HEADER_SIZE = 10

con = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    con.connect((address, port))
    print(f"Connected to server at {address}:{port}")

    while True:
        # GET USER INPUT
        message = input('Enter a message ("exit" to quit): ')

        # SEND MESSAGE TO SERVER
        header = f"{len(message):0{HEADER_SIZE}d}".encode("utf-8")
        con.send(header + message.encode("utf-8"))
        print(f"Sent to server: {message}")

        # RECEIVE ECHO FROM SERVER AND SAVE TO FILE
        received_message = ""
        msg_length = 0
        newmsg = True

        with open("client_received.txt", "w", encoding="utf-8") as f:
            while len(received_message) < msg_length:
                data = con.recv(BUF_SIZE)
                if not data:
                    print("Server disconnected")
                    break

                if newmsg:
                    msg_length = int(data[:HEADER_SIZE].decode("utf-8"))
                    chunk = data[HEADER_SIZE:].decode("utf-8")
                    newmsg = False
                else:
                    chunk = data.decode("utf-8")

                received_message += chunk
                f.write(chunk)  # Write to file as received

        print(f"Server echoed: {received_message}")
```

```
        print(f"Echo saved to: client_received.txt")

        if message == "exit":
            break

    except socket.error as e:
        print(f"Error: {e}")
    finally:
        con.close()
        print("Connection closed.")
```

Step-by-Step Example

Client sends: "Hello World"

Server Process:

1. Receives "Hello World" in chunks
2. Writes chunks to `server_received.txt` as they arrive
3. File content: "Hello World"
4. Reads complete file content: "Hello World"
5. Sends back: "0000000011Hello World"

Client Process:

1. Receives server response in chunks
2. Writes chunks to `client_received.txt` as they arrive
3. File content: "Hello World"
4. Displays: "Server echoed: Hello World"

File Operations Timeline

```
Time 1: Client sends "Hello World"
Time 2: Server receives chunk "Hello" → server_received.txt contains "Hello"
Time 3: Server receives chunk " World" → server_received.txt contains "Hello World"
Time 4: Server reads from file → gets "Hello World"
Time 5: Server sends back "Hello World"
Time 6: Client receives chunk "Hello" → client_received.txt contains "Hello"
Time 7: Client receives chunk " World" → client_received.txt contains "Hello World"
```

Advanced Features

1. File Timestamping

python

```python
import datetime

filename = f"message_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
with open(filename, "w", encoding="utf-8") as f:
    # Write message with timestamp
    f.write(f"[{datetime.datetime.now()}] {message}\n")
```

2. Message Logging

python

```python
# Append mode for logging all messages
with open("server_log.txt", "a", encoding="utf-8") as f:
    f.write(f"[{datetime.datetime.now()}] From {addr}: {message}\n")
```

3. Binary File Support

python

```python
# For binary files
with open("received_data.bin", "wb") as f:
    f.write(data)  # Write raw bytes

# Reading binary
with open("received_data.bin", "rb") as f:
    content = f.read()
```

Use Cases

1. **Message Persistence**: Save all communications to disk
2. **File Transfer**: Send files through the socket protocol
3. **Logging System**: Keep records of all messages
4. **Backup/Recovery**: Store messages for later retrieval
5. **Processing Pipeline**: Save → Process → Send back

Error Handling Considerations

python

```
try:
    with open("file.txt", "r") as f:
        content = f.read()
except FileNotFoundError:
    content = "File not found"
except PermissionError:
    content = "Permission denied"
except Exception as e:
    content = f"Error: {str(e)}"
```

This implementation provides a complete file-based echo system where messages are persisted to disk before being echoed back, useful for logging, persistence, and file processing applications.