University Of Science And Technology Of Hanoi

ICT Department

LAB REPORT

Title: Practical Work 1 – TCP File Transfer

Subject:	Distributed Systems (DS2026)
Student:	Le Nam Anh
StudentID:	23BI14029
Major:	Cyber Security

1 Introduction

The main objective of this practical work is to design and implement a simple one-to-one file transfer application operating over TCP/IP. Built upon the client-server model introduced in the lectures, the system functions through a command-line interface (CLI).

The system adheres to several key requirements:

- **Single Client-Server Pair:** The architecture consists of exactly one server process interacting with one client process.
- **Reliable Communication:** The system utilizes TCP sockets to ensure reliable, ordered byte-stream delivery between the endpoints.
- **File Transfer Capability:** The client is capable of sending files of arbitrary types and sizes to the server.

2 Protocol Design

2.1 Objectives

Our custom application-layer protocol was built to achieve:

- **Data Integrity:** Ensuring the file received is bit-for-bit identical to the source.
- **Stream Parsing:** Providing a mechanism (metadata header) for the server to distinguish between filename information and actual file content.
- **Simplicity:** Maintaining a lightweight structure that is easy to implement and debug.

2.2 Interaction Flow

The communication process is structured as follows:

1. **Handshake:** The client initiates a TCP connection to the server's open port.
2. **Metadata Transmission:** Before sending any file content, the client sends a structured header containing:
    - *Filename Length (4 bytes):* Tells the server how long the filename string is.
    - *Filename (N bytes):* The actual name of the file (UTF-8 encoded).
    - *File Size (8 bytes):* The total size of the file in bytes.
3. **Data Transmission:** The client streams the binary data of the file in 4KB chunks.
4. **Termination:** Once the data transfer matches the declared file size, the connection is closed.

2.3 Message Structure

The packet sent to the server is organized sequentially as shown in the table below:

Field	**Size**	**Description**
Name Length	4 bytes	Big-endian unsigned integer
Filename	Variable	UTF-8 string
File Size	8 bytes	Big-endian unsigned long long
Payload	Variable	Binary content of the file

Table 1: Protocol Message Structure

3 System Organization

3.1 Architecture

The system comprises two distinct scripts:

- `server.py`: Acts as the receiver. It binds to a specific port and waits for incoming requests. Upon connection, it decodes the header to set up the output file path, then writes the incoming byte stream to disk.
- `client.py`: Acts as the sender. It locates the target file, packages the necessary metadata using the struct library, and transmits the data over the network.

3.2 Operational Workflow

Server Side:

1. Initialize a TCP socket and bind it to 0.0.0.0.
2. Enter a listening state waiting for a client.
3. On connection, parse the 4-byte length, read the filename, then parse the 8-byte size.
4. Enter a loop to receive data chunks until the total bytes received matches the file size.

Client Side:

1. Connect to the server IP.
2. Calculate file size and encode the filename.
3. Send the packed header followed immediately by the file data.
4. Close the socket upon completion.

4 Implementation

The file transfer system is implemented in Python 3, using the built-in `socket` library for network communication and the `struct` module to pack/unpack binary values in network byte order.

4.1 Server-side Code

Listing 1: Server-side code (`server.py`) - Handles connections and saves files to the `received_files` directory.

```
import socket
import struct
import os
import sys

OUTPUT_DIR = "received_files"
BUFFER_SIZE = 4096

def start_server(port):
    if not os.path.exists(OUTPUT_DIR):
        os.makedirs(OUTPUT_DIR)

    server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    server_sock.bind(('0.0.0.0', port))
    server_sock.listen(1)
    print(f"Server listening on port {port}")

    try:
        while True:
            print("Waiting for a connection...")
            conn, addr = server_sock.accept()
            print(f"Connected from {addr}")

            try:
                raw_len = conn.recv(4)
                if not raw_len: break
                name_len = struct.unpack('!I', raw_len)[0]
                filename_bytes = conn.recv(name_len)
                filename = filename_bytes.decode('utf-8')
                filename = os.path.basename(filename)
                raw_size = conn.recv(8)
                filesize = struct.unpack('!Q', raw_size)[0]

                print(f"Receiving file: {filename} ({filesize} bytes)")
                output_path = os.path.join(OUTPUT_DIR, filename)
                received_bytes = 0

                with open(output_path, 'wb') as f:
                    while received_bytes < filesize:
                        chunk_size = min(BUFFER_SIZE, filesize - received_bytes)
                        data = conn.recv(chunk_size)
                        if not data: break
                        f.write(data)
                        received_bytes += len(data)
                print(f"File saved to {output_path}")

            except Exception as e:
                print(f"Error: {e}")
            finally:
                conn.close()
                print("Connection closed.")

    except KeyboardInterrupt:
        print("\nServer stopping...")
    finally:
        server_sock.close()

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python server.py <port>")
        sys.exit(1)

    port = int(sys.argv[1])
    start_server(port)
```

Listing 1: Server Code (server.py)

4.2 Client-side Code

Listing 2: Client-side code (`client.py`) - Connects to server and transmits file metadata and content.

```
import socket
import struct
import os
import sys

BUFFER_SIZE = 4096

def send_file(server_ip, server_port, filename):
    if not os.path.isfile(filename):
        print(f"File '{filename}' does not exist.")
        return

    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print(f"Connecting to {server_ip}:{server_port}...")
        sock.connect((server_ip, server_port))
        print("Connected.")

        file_basename = os.path.basename(filename)
        filename_bytes = file_basename.encode('utf-8')
        filesize = os.path.getsize(filename)

        sock.sendall(struct.pack('!I', len(filename_bytes)))
        sock.sendall(filename_bytes)
        sock.sendall(struct.pack('!Q', filesize))

        print(f"Sending file: {file_basename} ({filesize} bytes)")

        with open(filename, 'rb') as f:
            while True:
                chunk = f.read(BUFFER_SIZE)
                if not chunk:
                    break
                sock.sendall(chunk)

        print("File transmission complete.")

    except Exception as e:
        print(f"Error: {e}")
    finally:
        sock.close()

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python client.py <server_ip> <server_port> <
    filename>")
        sys.exit(1)

    server_ip = sys.argv[1]
    server_port = int(sys.argv[2])
    filename = sys.argv[3]

    send_file(server_ip, server_port, filename)
```

Listing 2: Client Code (client.py)

5 Testing and Results

The file transfer system was tested on a local machine using two terminal windows (one for the server and one for the client). Both client and server were run using Python 3.

Execution:

- Server command: `python server.py 5000`
- Client command: `python client.py 127.0.0.1 5000 test.txt`

Results: In a test run, a sample text file (`test.txt`, 263 bytes) was sent by the client. The server successfully received the metadata and stored the file in the designated `received_files/` folder. The content of the received file was verified to be identical to the original.

The protocol was further verified using larger files (such as images and PDFs) to ensure it works correctly for binary data and various file sizes. In all cases, the files were transmitted correctly without corruption.