University Of Science And Technology Of Hanoi
ICT Department

LAB REPORT

Title: Practical Work 1 – TCP File Transfer

Subject: Distributed Systems (DS2026)

Student: Nguyen Phuong Anh

StudentID: 22BA13020

Major: CS

1 Introduction

The goal of this practical work is to implement a simple **1–1 file transfer** application over TCP/IP in a command-line interface, based on the client–server model introduced in the lecture. The system must:

- Use exactly one server process and one client process.
- Use TCP sockets for reliable communication.
- Allow the client to send a file to the server.

In this report, I describe (i) how the application-level protocol is designed, (ii) how the overall system is organized, (iii) the main implementation details, and (iv) who did what in this work.

2 Protocol Design

The application-level protocol defines *what* is exchanged between the client and the server on top of TCP. We use a simple, custom protocol:

1. The client establishes a TCP connection to the server.
2. The client sends the length of the file name as a 4-byte unsigned integer in network byte order (big-endian).
3. The client sends the file name as a UTF-8 string.
4. The client sends the file size as an 8-byte unsigned integer.
5. The client sends the file content as a byte stream, split into chunks.
6. After receiving all bytes, the server stores the file on disk and sends a simple acknowledgment string (`"OK"`).
7. Both sides close the connection.

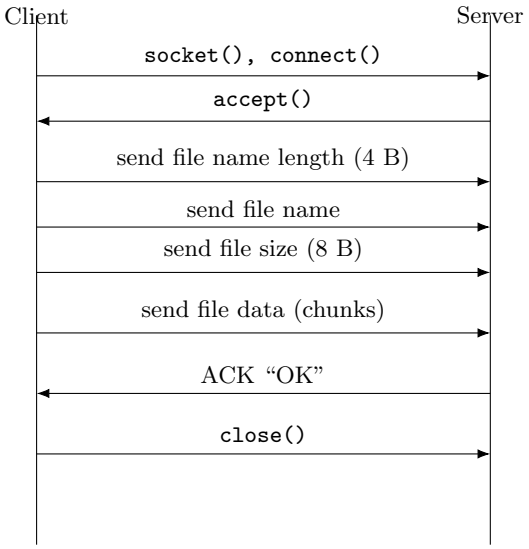

Figure 1: Application-level protocol for TCP file transfer.

3 System Organization

The system follows a classic client–server architecture. The overall organization is shown in Figure 2.

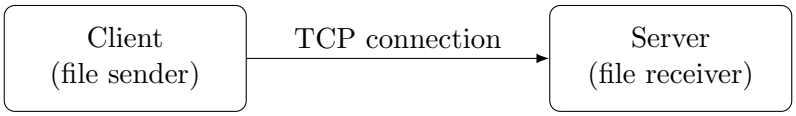

Figure 2: Overall client–server organization of the system.

Server

1. Create TCP socket, enable `SO_REUSEADDR`.
2. Bind to a fixed port and listen.
3. Accept one incoming connection.
4. Receive metadata and file data, write to disk.
5. Send acknowledgment and close the socket.

Client

1. Create TCP socket and connect to server.
2. Open local file; compute file name and file size.
3. Send metadata (name length, name, size).
4. Send file data by chunks until EOF.
5. Receive acknowledgment and close the socket.

4 Implementation of the File Transfer

The implementation is written in Python using the `socket` module.

Server-side snippet

Listing 1: Server-side handler for one client.

```
def recv_exact(conn, length):
    data = b""
    while len(data) < length:
        chunk = conn.recv(length - len(data))
        if not chunk:
            raise ConnectionError("Connection closed unexpectedly")
        data += chunk
    return data

def handle_client(conn, addr):
    print(f"[+] Connected from {addr}")

    raw = recv_exact(conn, 4)
```

```
    filename_len = struct.unpack("!I", raw)[0]

    filename_bytes = recv_exact(conn, filename_len)
    filename = filename_bytes.decode("utf-8")

    raw = recv_exact(conn, 8)
    filesize = struct.unpack("!Q", raw)[0]

    print(f"[+] Receiving file: {filename} ({filesize} bytes)")
    safe_filename = "received_" + os.path.basename(filename)

    received = 0
    with open(safe_filename, "wb") as f:
        while received < filesize:
            chunk = conn.recv(min(4096, filesize - received))
            if not chunk:
                raise ConnectionError("Connection closed while receiving
                    data")
            f.write(chunk)
            received += len(chunk)

    print(f"[+] File saved as {safe_filename} ({received} bytes)")
    conn.sendall(b"OK")
```

Client-side snippet

Listing 2: Client-side code to send a file.

```
filename = os.path.basename(file_path)
filesize = os.path.getsize(file_path)
filename_bytes = filename.encode("utf-8")

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((server_host, server_port))
    print("[+] Connected")

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

    print(f"[+] Sending file: {filename} ({filesize} bytes)")

    sent = 0
    with open(file_path, "rb") as f:
        while True:
            chunk = f.read(4096)
            if not chunk:
                break
            sock.sendall(chunk)
            sent += len(chunk)

    print(f"[+] File sent ({sent} bytes)")

    ack = sock.recv(1024)
    print(f"[+] Server replied: {ack.decode(errors='ignore')}")
```

5 Testing and Results

The implementation was tested on a single Windows machine using two terminals (PowerShell) and Python 3:

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

A sample text file of 30 bytes was transmitted. The server stored the file as `received_test.txt`; its content was identical to the original `test.txt`.

The same protocol also worked for larger files such as PDFs and images. The current implementation is single-threaded, therefore it can serve only one client at a time, which is acceptable for this practical work.