

TCP File Transfer Protocol

Distributed Systems Practical

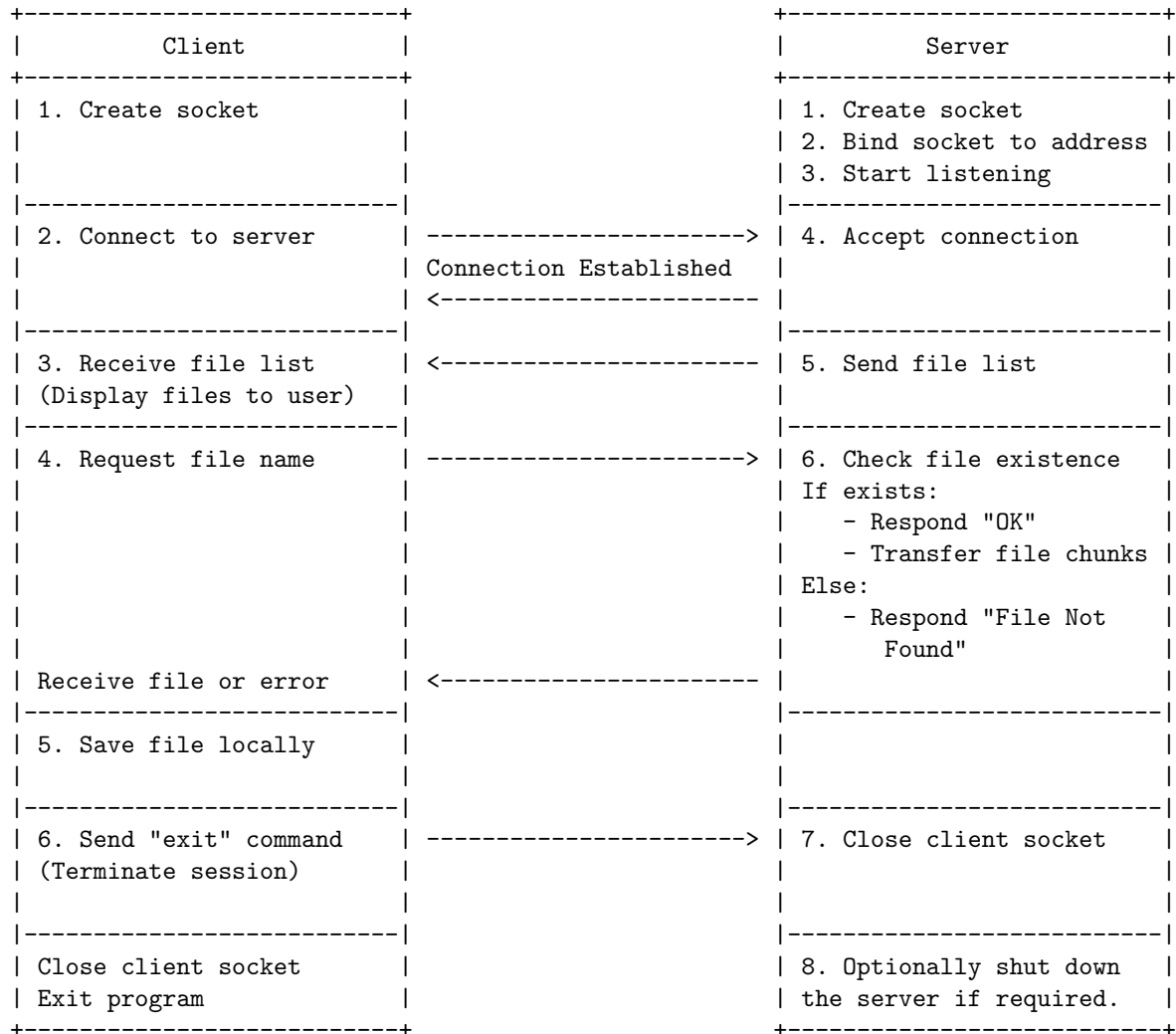
1 Protocol Design

Our TCP-based file transfer protocol is designed to allow clients to:

- List available files on the server.
- Request specific files for download.
- Download files reliably using TCP.
- Exit the connection gracefully, signaling the server to shut down if needed.

1.1

Protocol Design Diagram

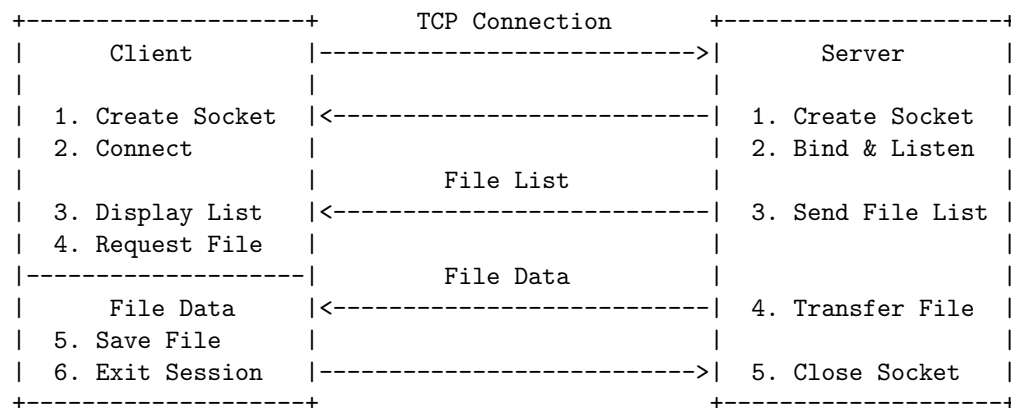


2 System Organization

The system consists of two components:

1. **Server:** Hosts the files and serves client requests. Runs continuously and manages multiple clients using threads.
2. **Client:** Connects to the server, lists available files, downloads files, and exits when done.

2.1 System Organization Diagram



3 File Transfer Implementation

The server and client are implemented using Python's `socket` library. Code snippets of the implementation are shown below.

3.1 Server Code

```

1 def handle_client(client_socket, client_address):
2     files = os.listdir(SERVER_DIRECTORY)
3     file_list = "\n".join(files)
4     client_socket.send(file_list.encode())
5
6     while True:
7         requested_file = client_socket.recv(BUFFER_SIZE).decode().strip()
8         if requested_file.lower() == 'exit':
9             break
10
11         file_path = os.path.join(SERVER_DIRECTORY, requested_file)
12         if os.path.exists(file_path):
13             client_socket.send("OK".encode())
14             with open(file_path, 'rb') as f:
15                 while chunk := f.read(BUFFER_SIZE):
16                     client_socket.send(chunk)
17         else:
18             client_socket.send("ERROR: File not found.".encode())
19
20     client_socket.close()

```

Listing 1: Server File Handling

3.2 Client Code

```
1 def start_client():
2     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3     client_socket.connect((SERVER_HOST, SERVER_PORT))
4
5     file_list = client_socket.recv(BUFFER_SIZE).decode()
6     print("Available files:\n" + file_list)
7
8     while True:
9         requested_file = input("Enter the file name to download (or 'exit' to quit): ").
10            strip()
11         client_socket.send(requested_file.encode())
12         if requested_file.lower() == 'exit':
13             break
14
15         response = client_socket.recv(BUFFER_SIZE).decode()
16         if response == "OK":
17             file_path = os.path.join(CLIENT_DIRECTORY, requested_file)
18             with open(file_path, 'wb') as f:
19                 while chunk := client_socket.recv(BUFFER_SIZE):
20                     f.write(chunk)
21         else:
22             print(response)
23     client_socket.close()
```

Listing 2: Client File Request

4 Responsibilities

- **Server:** Handles client connections, provides the list of files, and sends requested files.
- **Client:** Lists available files, requests files for download, and manages local storage.