USTH Master Spoiler Alert

01.tcp.file.transfer

Nguyen Quynh Trang – 22BA13303

1 Protocol Design

This practical work focused on the construction of a basic distributed system for file transfer. The goal was achieved by developing a Client-Server architecture based on standard C sockets. To manage data flow efficiently, a bespoke, lightweight **two-phase** application protocol was designed on top of TCP. This approach capitalizes on TCP's built-in mechanisms for sequencing and error correction, guaranteeing the reliable and ordered transfer of file data.

1.1 Network Protocol Selection

TCP/IP: We chose TCP/IP for its connection-oriented nature (`SOCK_STREAM`), which automatically handles sequencing, error checking, and retransmission. This ensures the file is delivered reliably without needing to build complex fault-handling logic at the application layer.

1.2 Data Exchange Flow

The session follows a structured sequence to separate filename from the main data stream:

Step	Initiated by	Data Transferred	Purpose
1.Connection Setup	Client to Server	TCP SYN request	Establish a reliable full-duplex connection.
2.Filename Transmission	Client to Server	File name string	Server uses first `recv()` to obtain filename and create output file.
3.Content Transfer Loop	Client to Server	File content blocks	Efficient streaming of the full file.
4.Session Termination	Client to Server	`close()`	Generates EOF signal indicating no more data will be sent.
5.Server Closure	Server	EOF (`recv()` returns 0)	Server finalizes file write and closes socket.

2 System Organization

The system utilizes the classic Client–Server architecture, where the Server passively waits for connections and the Client actively initiates the session.

2.1 Socket Flow Diagram

The organization strictly adheres to the standard TCP session flow defined in the course material, as visualized in Figure 1. This diagram clarifies the sequence of socket operations necessary to establish, maintain, and terminate a reliable connection.

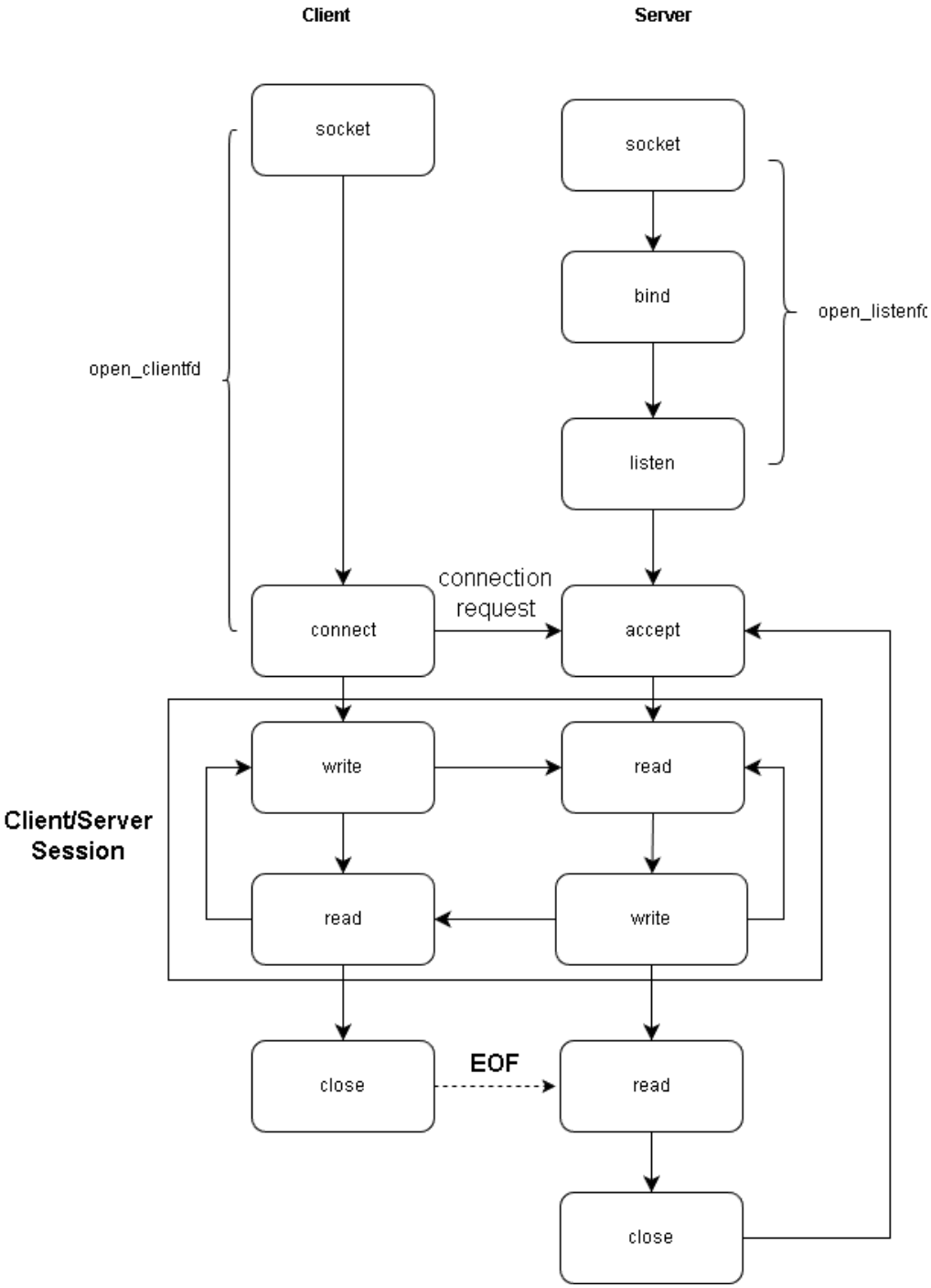

Figure 1: Socket Flow Diagram for TCP File Transfer Session

The system architecture confirms the sequential interaction between the Client and the Server in four main phases: Listener Setup, Connection Establishment, Data Exchange (write/read), and Termination (close/EOF).

- **Phase 1: Listener Setup (open_listenfd):** The Server initializes its socket, binds it to the fixed port 8080, and calls `listen()`, essentially putting itself in a waiting room for incoming connections.

- **Phase 2: Connection Establishment:** The Client creates its socket and sends a connection request using `connect()`. The Server grabs this request, verifies it, and returns a new connection-specific file descriptor through `accept()`, establishing the link.
- **Phase 3: Data Session & Disk I/O:** This is the actual file transfer phase. The Client starts by `write`ing the filename first (our simple protocol), followed by continuous `write` operations for file content blocks. The Server continuously `read`s this stream from the network and writes the data to the local file (disk I/O).
- **Phase 4: Termination (close/EOF):** Once the Client finishes sending the file, it calls `close()`. This action sends the **End-of-File (EOF)** signal to the Server, allowing the Server to detect the completion of the transfer when its next read returns 0. Both sides then safely close their respective sockets.

This flow diagram is essential as it confirms the synchronization required at the transport layer for our application protocol to function reliably.

2.2 Configuration Details

Component	Role	Key Functions	Configuration
Server	Passive Listener / File Receiver	`bind()`, `listen()`, `accept()`, `recv()`	IP: `INADDR_ANY`; Port: 8080
Client	Active Initiator / File Sender	`connect()`, `send()`, `gethostbyname()`	IP via `gethostbyname`; Port: 8080

3 File Transfer Implementation

3.1 Code Snippet: Client Sender Logic

The client's primary responsibility is to open the source file, send the file name once, and then continuously stream the content using a buffer.

```
// CLIENT: Sends filename and then enters the content loop

// 1. Send filename (First step of the application protocol)
send(client_fd, filename_to_send, strlen(filename_to_send), 0);

// 2. Loop to read file from disk and send data through the socket
while ((bytes_read = read(file_fd_read, data_buffer, BUFFER_SIZE)) > 0) {
```

```
    if (send(client_fd, data_buffer, bytes_read, 0) < 0) {
        perror("ERROR sending file data");
        break; // Exit loop on send failure
    }
}
// Close ensures the EOF signal is sent to the Server
close(file_fd_read);
close(client_fd); // Closes the connection and sends EOF
```

Listing 1: Client Code Snippet: Sender Logic

3.2 Code Snippet: Server Receiver Logic

The server handles receiving the filename, creating the target file, and persistently reading from the socket until the EOF signal is received (i.e., recv() returns 0).

```
// SERVER: Receives filename and then enters the content loop

// 1. Receive the filename first
bytes_received = recv(client_socket, client_filename, BUFFER_SIZE - 1, 0);

// 2. Open local file (saved_filename)

// 3. Loop to receive content and write to the local file
while ((bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0)) > 0) {
    if (write(file_fd_write, buffer, bytes_received) < 0) {
        perror("ERROR writing to file");
        break; // Exit loop on write failure
    }
}

// bytes_received == 0 indicates successful EOF reception.
close(file_fd_write);
close(client_socket); // Clean up the connection
```

Listing 2: Server Code Snippet: Receiver Logic

3.3 Error Handling

All system calls are immediately followed by error checks using `perror()` to output precise system-level error messages (based on `errno`), which is critical for debugging distributed systems.

4 Work Distribution

This practical work was executed individually. Roles were distributed as follows:

- **Server Development:** Socket setup, IP/port resolution, filename handling, file reception loop.
- **Client Development:** Host resolution (`gethostbyname`), connection setup (`connect`), file transmission loop.
- **Reporting & Testing:** Protocol design, integration testing, report writing.