Assignment 2: UDP Sockets

Deadline: January 27, 2025 EOD

SUBMISSION INSTRUCTIONS

Prepare a detailed observation and analysis report and the appropriate screenshots from the Wireshark and Terminal for listed questions with specific details asked in individual tasks. Submit your report in PDF format with <NAME>_<ROLL NO>_Report.pdf. The pcap file should be uploaded to a public Google drive folder and the link needs to be given within the code. Zip all these files, include the source code of the two programs into a single zip file <NAME>_<ROLL NO>.zip and submit on MS Teams.

NOTES:

1. You should mention your name (as per ERP), roll number and the Assignment number within the comment line at the beginning of each program. A sample header will look as follows.
   ```
   =====================================
   Assignment 2 Submission
   Name: <Your_Name>
   Roll number: <Your_Roll_Number>
   Link of the pcap file: <Google_Drive_Link_of_the_pcap_File>
   =====================================
   ```
2. The code should have proper documentation and indentation. Unreadable codes will not be evaluated.
3. The code should get executed to the lab machines. If we get a compilation error or runtime error during executing your code on the lab machines, appropriate marks will be deducted.
4. Any form of plagiarism will incur severe penalties. You should not copy the code from any sources. You may consult online sources or your friend, but at the end, you should type the program yourself. We may ask you to explain the code, and if you fail to do so, you'll be awarded zero marks.

Objective:

The objective of this assignment is to get familiar with datagram sockets using POSIX C programming and to analyze the behavior of the communication between client and server using **Wireshark**. The target is to establish communication between two processes (client and server) using a datagram socket and observe the network traffic using Wireshark.

Problem Statement:

In this assignment, you need to write the code (using POSIX C) for a server and a client application that will use the datagram socket (UDP protocol at the transport layer) to communicate among themselves. As we discussed in the theory classes, UDP provides unreliable, connectionless datagram delivery across two processes. In your application program, the server stores a set of files (you may consider three different files for your implementation). The files contain a set of alphanumeric words, one at each line. However, the first word in each of the files must be `HELLO`, and the last word must be `FINISH`. We assume that HELLO and FINISH are two special keywords that do not come anywhere except at the first line (HELLO) and at the last line (FINISH).

Here is the content of a sample file:

```
HELLO
CS31206
CS39006
CS31208
FINISH
```

The files are the text files. The file names would be simple like <Your_Roll_Number>_File1.text. Do not use any special character on the file name or file text.

The interaction between the client and the server follows these steps:

1. The **client** first sends the file name to the **server**.
2. On receiving the file name, the **server** checks if the file exists in the local directory:
    a. If the file is not available, the server sends a `NOTFOUND <FILENAME>` message to the client, and the client displays an error “FILE NOT FOUND” and exits.
    b. If the file exists, the server reads the first line (`HELLO`) and sends it to the client.
3. Upon receiving `HELLO`, the **client** creates a new file to store the received words. It sends $WORD_i$ messages to the server, starting with $WORD_1$ to request the first word.
4. For each $WORD_i$ request, the **server** sends the corresponding word from the file to the **client**. This continues until the server sends the last word (`FINISH`).
5. Upon receiving the `FINISH` message, the **client** writes it to the file, closes the file, and exits. The **server** also terminates.

Part-1: Socket Programming

You need to implement two programs.

1. **`wordserver.c`** (server program): The server listens for a file request from the client, retrieves the file contents, and sends them back to the client. *Marks:15*
2. **`wordclient.c`** (client program): The client sends a file name to the server, receives the file contents word by word, and writes them to a new file. *Marks:20*

Note:

- ➢ Ensure proper handling of messages (`HELLO`, $WORD_i$, `FINISH`, and `NOTFOUND`).
- ➢ Ensure that the content of the received file matches the original file exactly.

Part-2: Wireshark Analysis

Use **Wireshark** to capture and analyze the communication between the client and server. Perform the following:

1. Capture all packets exchanged between the client and server during execution. Show the screenshots. *Marks:2*
2. What protocol is used for communication? *Marks:1*
3. What are the source and destination IP addresses and ports? *Marks:2*
4. What is the size (in bytes) of the `FILENAME` request sent by the client? *Marks:1*
5. What is the size of the server's response for `HELLO` and the first word ($WORD_1$)? *Marks:2*
6. Inspect the payload of packets where the words are transmitted. Show the UDP payloads of those packets. *Marks:2*
7. Measure the total time taken for the file transfer from start to finish. *Marks:3*
8. What is the average size of each packet during the communication? Marks:2