COP701: Assignment #3

September 28, 2023

1 Problem statement

The Trivial File Transfer Protocol (TFTP) is a simple protocol that provides basic file transfer functionality with no user authentication. Implement a TFTP server from scratch that supports both getting and putting files between the client and the server. You are supposed to implement both the server and the client and demonstrate their working together. You need to follow the specifications given below.

1.1 TFTP server

The specification of TFTP can be found in the RFC 1350. Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened, and the file is sent in fixed-length blocks of 512 bytes. Each data packet contains one block of data and must be acknowledged by an acknowledgment packet before the next packet can be sent. A data packet of less than 512 bytes signals the termination of a transfer. Your implementation should:

- Use socket programming to listen on the well-known port 69 for incoming connections.
- Use multi-threading to cater to multiple clients in parallel.
- Support all of the following packet types defined in the TFTP RFC 1350:
    1. Read request (RRQ)
    2. Write request (WRQ)
    3. Data (DATA)
    4. Acknowledgment (ACK)
    5. Error (ERROR)
- Provide valid error messages along with the corresponding standard error codes in the ERROR packets.
- Have appropriate timeouts and acknowledgments for the packets.

1.2 TFTP client

You should also implement a custom client program that takes input commands from the user (argv) and communicates with your TFTP server following the appropriate protocol. Similar to the TFTP server, your client should support all packet types, splitting the file into data blocks of 512 bytes. (NOTE: Ensure that you do not duplicate the code here and implement them as libraries that are included.)

In addition to that, the client is supposed to compress the given file on a WRITE request before transferring the file to the server. Similarly, the client should decompress the file on receiving it from the server and save the decompressed file for read requests. You need to implement the logic for compression and decompression on your own. The details of the algorithm to be used are discussed in Section 1.3

The client program can be invoked in the following manner:

1. Read a file from the TFTP server:
   ./client READ filename.txt

2. Write a file to the TFTP server:
   ./client WRITE filename.txt

1.3 Compression algorithm

Compression is the process of encoding information using fewer bits than the original representation. Hence it reduces the file size using a compression algorithm. Compression algorithms are divided into two broad categories: lossless and lossy compression. The Deflate algorithm is a popular lossless compression algorithm used in zip, zlib and gzip. It uses a combination of LZ277 and Huffman coding where deflate is the process of compression and inflate is the process of decompression.

You are required to implement a modified version of the Deflate algorithm using the in-place linear time version of the Huffman coding algorithm mentioned in Section 2.4 of [1] (Read the algorithm from the paper and understand how it is different from the standard Huffman coding algorithm). You can choose any of the following methods for your implementation of the compression/decompression algorithm. For the sake of simplicity, you can assume that we will only be compressing text files.

1. Modify the code in the open-source zlib library implementation with the custom Huffman algorithm and call this library from your TFTP client code. This is the recommended method.

2. Implement the custom algorithm from scratch as part of the TFTP client.

1.4 Bonus

You are encouraged to consider one or more of the following features for extra credit.

- Make the TFTP server compatible for usage with any standard TFTP client in addition to your client.
- TFTP is insecure unlike FTP. Consider adding an authentication layer for the user and/or encryption for the data being transferred (For example: using TLS).
- TFTP lacks most of the features of a regular FTP. You can consider any of the additional features given by FTP that are not part of the TFTP RFC 1350.

2 Logistics

- The **deadline** for this assignment is **12/11/2023 at 11:59 PM**. It is a hard deadline and will not be extended.
- This assignment can be done in a group of two people. Only one of you must submit. (40 Marks)
- You can only use C/C++ in this assignment. You are not supposed to use any other programming language.
- You need to create a private git repository either on https://git.iitd.ac.in or github. Git commit history will be checked during evaluation.
- ANY form of **plagiarism** will not be tolerated.
- Submission will be made on Moodle. You need to submit all your code and a pdf format report. Compress all these in a tar file with the name $< entry\_number1 >$ _ $< entry\_number2 >$ *.tar* and upload on Moodle.
- You will be graded on the output of your code (working of the TFTP server and the client along with compression), the coding style and your viva/presentation.
- Any doubts regarding the course/assignment should be asked on Piazza.

2.1 Marks distribution

Report	10%
Coding style	10%
Unit tests	10%
Regular git commits (from both the members)	5%
Compression and decompression	25%
TFTP server and client	40%

References

[1] A. Moffat, “Huffman coding,” *ACM Comput. Surv.*, vol. 52, no. 4, aug 2019. [Online]. Available: https://doi.org/10.1145/3342555