TFTP server and client side implementation

Sviatoslav Shishnev

November 20, 2023

Contents

1 Introduction

The goal of this project is to implement a pair of programs that facilitate communication using the Trivial File Transfer Protocol (TFTP). This protocol, outlined in RFC 1350, serves as a lightweight means for transferring files between devices in a network. The project comprises a TFTP client and server, each equipped with distinct functionalities and capabilities.

1.1 Client Features

The TFTP client is designed to operate primarily in octet mode, adhering to the specifications defined in RFC 1350. It has the capability to establish communication through Read Request (RRQ) and Write Request (WRQ) queries, allowing for the retrieval and transmission of files. The octet mode ensures efficient binary data transfer.

1.2 Server Features

The TFTP server extends its functionality to support multiple modes of operation, including netascii and octet. In addition to RFC 1350 compliance, the server implements advanced features outlined in RFC 2347, RFC 2348, and RFC 2349.

1.2.1 Option Negotiation

The server engages in a negotiation process to support dynamic configuration options:

- **Blocksize Option:** Allows the adjustment of block size for data transfers.
- **Timeout Option:** Enables the specification of timeout intervals for reliable communication.
- **Transfer Size Option:** The Transfer Size Option is a pivotal feature that enhances the reliability and efficiency of file transfers in our TFTP server. This option provides essential information about the size of the file being transferred, allowing the server to make informed decisions during the transmission process.

2 Application Overview

The TFTP application consists of both a client and a server component.

2.1 Client

The client is a synchronous program designed for active communication with the server. It accepts the server's IP address and port as command-line arguments. The client supports the octet mode and is compliant with RFC 1350 standards for Read Request (RRQ) and Write Request (WRQ) queries.

2.2 Server

The server operates asynchronously and utilizes multiple processes for concurrent communication. It does not rely on external libraries for asynchronous support. Instead, it creates additional processes, each assigned a new socket with the same address and a dynamically assigned port. Each client communicates with a distinct process on a different port.

The server do not support any options defined in RFC 2347

3 Communication Overview

The communication process begins with either a Read Request (RRQ) or Write Request (WRQ) from the client.

3.1 Read Request (RRQ)

Upon receiving an RRQ, the server checks for specified options, as defined in RFC 2347. If options are present, they are parsed, and the server confirms negotiations by sending an Options Acknowledgment (OACK) packet.

After confirming negotiations, the server waits for an acknowledgment (ACK) packet with a block number of 0. Upon receiving the ACK, the server sends the first Data packet. If no options are provided, the Data packet is sent without the OACK step.

The server then waits for an ACK packet, and if the ACK is not received within the timeout period (default or specified by option), the server retransmits the Data packet. This process is repeated up to three times. If no ACK is received after three retransmissions, the server terminates the communication. If the ACK is successfully received, the communication ends when the size of the sent Data packet is below the standard block size or the block size provided by the option.

3.2 Write Request (WRQ)

Upon receiving a Write Request (WRQ), the server checks for options. If options are present, they are parsed, and negotiations are confirmed by sending an Options Acknowledgment (OACK) packet.

Then server waits to receive the first Data packet containing the data to be written. Upon successfully receiving the Data packet, the server sends an acknowledgment (ACK) packet to the client.

Subsequently, the server continues to send ACK packets for each received Data packet. If the next Data packet is not received within the timeout period (default or specified by option), the server retransmits the ACK packet. This process is repeated up to three times. If no Data packet is received after three retransmissions, the server terminates the communication.

In the case of successful receipt of the Data packet, the communication ends when the size of the received Data packet is below the standard block size or the block size provided by the option.

4 Conclusion

In conclusion, the implementation of the Trivial File Transfer Protocol (TFTP) client and server has been successfully achieved. The client supports communication using the octet mode and adheres to the specifications outlined in RFC 1350 for Read Request (RRQ) and Write Request (WRQ) queries.

On the server side, asynchronous processing has been incorporated to handle multiple clients simultaneously. The server supports octet mode. Notably, netascii mode and options defined in RFC 2347, RFC 2348, and RFC 2349 are not supported in this implementation.

Throughout the development, references to key RFC documents, namely RFC 1350, RFC 2347, RFC 2348, and RFC 2349, were consulted to fulfill their requirements .

This project has provided valuable insights into TFTP, network protocols, asynchronous programming, socket programming. It's important to recognize the areas where improvements can be made, such as the lack of support for certain modes and options.

As a personal note, the project also serves as a reminder of the importance of effective time management.

5 References

- RFC 1350: Trivial File Transfer Protocol (TFTP)
  - `https://datatracker.ietf.org/doc/html/rfc1350`
- RFC 2347: TFTP Option Extension
  - `https://datatracker.ietf.org/doc/html/rfc2347`
- RFC 2348: TFTP Blocksize Option
  - `https://datatracker.ietf.org/doc/html/rfc2348`
- RFC 2349: TFTP Timeout Interval and Transfer Size Options
  - `https://datatracker.ietf.org/doc/html/rfc2349`