VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Network Management and Network Administration

Project – TFTP Client

Author: Lukáš Chmelo

Login: xchmel33

Contents

About TFTP – Trivial file transfer protocol

Introduction

A simple protocol used to transfer files from/to remote server, but unlike it's sibling FTP it cannot list directories and has no user authentication. However, implementation is done on top of UDP and is much simpler.

File transfer

Transfer begins with a request to read/write a file, which creates a connection and sever either grands or denies request depending on permissions, file existence etc. If server grands read request client receives data packet with block number 0 containing the first 512 bytes block of data. In the next step this data packet is acknowledged by client and next data packet can be received. On the other hand, if the server is granting write request, that means client received an acknowledgement packet with block number 0 and data packets can be send to the server. Server than sends acknowledgement packets acknowledging the received data packets. Transfer ends by receiving or sending a packet shorter than 512 bytes, which is indicating end of file.

Modes of transfer

In modern implementations, two modes of transfer are supported: 'netascii'(ascii) and 'octet'(binary). The 'netascii' mode sends characters during transfer while 'octet', which is the default mode, sends raw bytes of data. There is also mail mode, but it is "obsolete and shouldn't be used" [1].

Packets

TFTP uses five types of packets of different sizes. Header of each packet starts with a 2 bytes opcode:

1. RRQ – read request
2. WRQ – write request
3. DATA – data packet
4. ACK – acknowledgement packet
5. ERROR – error packet

Read and write request packets are very similar. After opcode, their header continues with string filename, then 1 byte for termination of string filename, then string for mode and final byte for terminating string mode.

In general, acknowledgement and data packets are linked by a common block number, starting from 1. Acknowledgement and data packet consists of 2 bytes for opcode and 2 bytes for block number, but data packet has also additional space for n bytes of data.

Error packet causes premature termination of transfer. It consists of 2 opcode, 2 bytes for error code and string error message terminated by a zero byte. It also serves as acknowledgement for any other packet.

Client Implementation

Main loop

Program is run typing `./mytftpclient` It starts by looping inside main function generating > and getting arguments from current line. Arguments are parsed into program options by using function `get_options`. If any application function returns error, application shows errors but doesn't terminate. Application can be terminated by typing command `exit`.

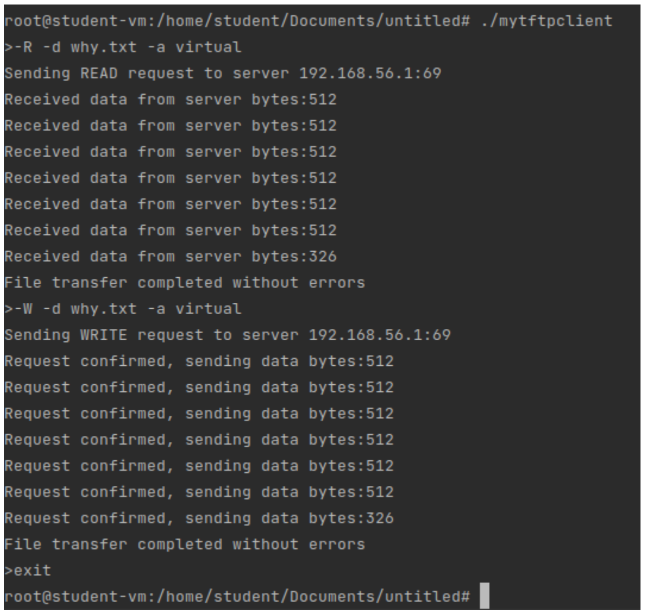
Program options

Function `get_options` firstly converts line of arguments into array. Options array is set to default values as specified in readme.txt. Array of arguments is parsed into options array with or without argument which is the next element inside argument array. Options that were not received remain default. After parsing, final check is done if mandatory options were provided and options with arguments are valid. In case of error, empty array is returned into main.

TFTP client

Function `TFTP_client` ensures file transfer onto a remote server. It checks if server address is valid and binds client address with server address which is later used in communication. Variable buffer is used as memory buffer for packets. RRQ/WRQ packet is created with corresponding opcode, filename and mode on pointer p pointing to memory buffer's 3rd byte. Request is then sent to server and depending on the mode either data or acknowledgement packet is received. In case of error current transfer is terminated. Functions keeps looping receiving and sending packet while short packet arrives or is sent, which completes the transfer and next transfer can begin,

Testing



```
root@student-vm:/home/student/Documents/untitled# ./mytftpclient
>-R -d why.txt -a virtual
Sending READ request to server 192.168.56.1:69
Received data from server bytes:512
Received data from server bytes:512
Received data from server bytes:512
Received data from server bytes:512
Received data from server bytes:512
Received data from server bytes:512
Received data from server bytes:326
File transfer completed without errors
>-W -d why.txt -a virtual
Sending WRITE request to server 192.168.56.1:69
Request confirmed, sending data bytes:512
Request confirmed, sending data bytes:512
Request confirmed, sending data bytes:512
Request confirmed, sending data bytes:512
Request confirmed, sending data bytes:512
Request confirmed, sending data bytes:512
Request confirmed, sending data bytes:326
File transfer completed without errors
>exit
root@student-vm:/home/student/Documents/untitled#
```

Conclusion

Project was unfinished because of time constrains. I have had many problems with C++, but improved my skills in programing in this language. The project wasn't very hard over all, but I got stuck for days trying to run virtual machine and that cost me a lot of time so I wasn't able to implement all required functionality.

References

[1] K. Sollins. RFC 1350, Network Working Group MIT, July 1992