

TFTP Lab – Socket Programming in C

Assignment in Computer Networks II
Department of Information Technology, Uppsala University

1 Formalities

This assignment needs to be performed in groups of two. You are encouraged to use the provided resources and to discuss with other groups. However, you are not allowed to use code from other groups doing the assignment. If you discuss with another group, please state so in the questionnaire form (see hand in procedure for details) to give them credit and avoid suspicion of cheating. The assignment can give you either 0, 2 or 3 points depending on how well your solution works. See the section on grading for more details on scoring.

There are two lab session dedicated to this lab, we highly recommend that you come prepared, make use of them as much as possible and ask questions.

2 Motivation

'After completing this lab you should be able to implement and make design choices regarding the basics of socket programming in the form of a functioning TFTP client.'

In the course description of Computer Networks 2, one of the stated learning outcomes are "Design and implement communicating applications". This lab is meant to train you in exactly that. The purpose of this assignment is that you get hands on experience and practice network programming, specifically socket programming.

3 Assignment

The assignment is made in an open way with no step-by-step instructions. You should be able to take in all the necessary information to plan and design your work and solution. Below follows information that is essential in completing this lab.

3.1 Introduction

This lab involves writing a TFTP client in C. TFTP stands for Trivial File Transfer Protocol and is the predecessor to FTP. It is still used today for transferring new firmware to routers and other embedded devices and also for booting network terminals. It is a simple protocol and therefore straightforward to understand. It has two main uses:

- Transferring a file from a client to server.
- Transferring a file from a server to a client.

Your task will be to write code for a client (you do not have to implement a server) that can perform both of these functions. In other words, you need to write code that can *PUT* a file from your client onto a server and *GET* a file from a server to your client.

3.2 How to Proceed

First

The first thing you need to do after finishing reading this instruction is to very thoroughly read the RFC1350 that details TFTP. The RFC document contains all the information you need about how TFTP works, so it is essential that you read and understand it before starting to program.

Second

Next step is to download the skeleton code from the lab homepage. It contains the following three files:

- `tftp.c` - *This is the only file that you should edit. Look through it and read the comments. Note that `/ * ... * /` marks places where you should insert code.*
- `tftp.h` - *This files contains all the predefined variables and data structures that you need, you should use them throughout your code. Note that some names match what you have read in the RFC. Do not add anything in this file since you will not hand it in. But you need to edit the port number to change between ports when testing.*
- `Makefile` - *This file is used to compile your program using `make`, do not edit it.*

Familiarize yourself with them, they are an essential help in completing the assignment.

Third

Read up on C syntax, especially on how pointers work if you do not already know. Remember that a pointer is a memory address, recall how to work with them. Also read up on socket programming and what functions there are that you can use. Some links are provided on the lab homepage.

Programming

Now you are ready to start programming the client that should be able to GET a file from a server as well as PUT a file onto a server.

You will need to extend the skeleton code in `tftp.c` and do the following

- Write the socket handling code to open the socket to the server (code for closing the socket is provided).
- Open and close the file being dealt with.
- Construct packets using the pre-defined packet header and send the following
 - Read request packets
 - Write request packets
 - Acknowledgment packets
 - Data packets
- Handle receiving data and perform the appropriate action
 - Write data to file
 - Read data from file and send to server
 - Handle errors from server

Compiling

The skeleton code already contains a Makefile for building. Run the program `make` to compile your code. Compiler warnings: The code is compiled with the `-Wall` flag, your code needs to compile without any warnings when you hand it in.

Testing and debugging

One of the major tasks in this assignment is actually testing and debugging your code. We highly recommend you to do all your testing on the schools UNIX system since it is there that the code will be evaluated by us.

For testing, there is a TFTP server running on the machine `joshua.it.uu.se` which is accessible inside the university network. You can access it using different port numbers to test different behavior. To change the port number that your tftp program uses change the `#define TFTP_PORT 69` in `tftp.h` to one of the following.

The following ports are open

- 69 - The standard port, reserved for TFTP. Only accessible inside the university network.
- 6969 - Same as 69 but accessible outside the university network.
- 10069 - Generates packet loss, you need to send a large enough file to get some packet loss. (Accessible outside the university network)
- 20069 - Generates duplicates, you need to send a large enough file to get duplicates. (Accessible outside the university network)

If you wish to retrieve a file, there are 3 test files available on the server: small.txt, medium.pdf, large.jpeg. If you wish to upload something feel free. However, do not upload objectionable material as all TFTP traffic is logged and offenders will be reported. The university Acceptable Usage Policy (AUP) is available [here](#).

To run the program and get the file small.txt from the server joshua.it.uu.se simply type (this will of course not work until you have implemented what is necessary)

```
./tftp -g small.txt joshua.it.uu.se
```

Some more helpful tips when testing your program.

File size The first thing to check when completing a transfer is that the file size of the original and the transferred file are exactly the same. The filesize of the provided files are the following.

small.txt	1931 bytes
medium.pdf	17577 bytes
large.jpeg	82142 bytes

MD5 Verify that the md5 of your transferred file is the same as the original. If two files have the same md5 they are exactly the same. The Unix command is md5sum. The md5 of the provided files are the following.

small.txt	667ff61c0d573502e482efa85b468f1f
medium.pdf	ee98d0524433e2ca4c0c1e05685171a7
large.jpeg	f5b558fe29913cc599161bafec08ccf

Wireshark A very good way to debug your code and, especially that you format the packets correctly, is to use Wireshark network protocol analyzer (or other such programs). However, due to privacy and user policy issues it is not available on the UNIX system, so if you want to use it you need to download it to your own laptop. A link can be found on the lab homepage. Run Wireshark and capture traffic while running your tftp client. Select a captured UDP packet and right-click and choose "Decode as" then select TFTP. That way you can see how every packet you send out "actually" looks on the network as well as read the TFTP related info. If Wireshark reports "malformed packet" for the packets that you have sent out, then you are doing something wrong. Often it is a case of a trailing zero.

3.3 Grading

First of all your program must fulfill the following requirements:

- Your code has to compile without warnings using the provided and unaltered tftp.h and Makefile.
- Your program must not crash.
- Your program must perform the correct network/host byte order conversions.
- Your program must send the correct length of packets.
- Your program must be able to perform error handling. (That is, if the TFTP server returns an error message, your program must be able to fail in the correct way as specified in the RFC. A simple way to test is to ask for a file that does not exist.)

On this lab you can get up to 3 points, for more info on course grading see the course info page. You get:

- 2 points if your program works and it can correctly send a file to the server and correctly receive a file from the server. This is verified by successful transfer using port 69. If only minor mistakes were made in this part you could potential still get 1 point (only applicable in rare cases).
- 1 additional point if you also can handle lost and duplicate packets. This is verified by successful transfer using port 10069 and 20069.

The correction and grading of your assignment will take place on hamberg.it.uu.se, so make sure it works properly on the school UNIX system, especially if you are working on the assignment using your own laptop. We will test getting and putting files and verify that the files are exactly the same (using file size and md5) as the original files.

3.4 Hand in Procedure and Deadline

When you have completed the assignment you need to complete and fill out questionnaire.txt. The questionnaire is used as a help for you to know what we will be testing and that you see how well your solution works. Keep in mind that it takes a bit of time to fill it out since you are supposed to run your code using different configurations. After filling out the questionnaire, using your final version of the code, you will have a good idea of how many points your solution may get.

For details on deadline and where to submit see the lab homepage.