1 Help along the way

This document contains some extra information that may help you to familiarize yourself with the task and give some suggestions on how to plan the work. The information here is not necessary to complete the assignment and perhaps not useful to everyone. Read it if you want some more help along the way.

1.1 Links

- Socket programming `http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html`
- `socket` (function) `http://www.kernel.org/doc/man-pages/online/pages/man2/socket.2.html`
- `close` (function) `http://www.kernel.org/doc/man-pages/online/pages/man2/close.2.html`
- `sendto` (function) `http://www.kernel.org/doc/man-pages/online/pages/man2/sendto.2.html`
- `recvfrom` (function) `http://www.kernel.org/doc/man-pages/online/pages/man2/recv.2.html`
- `select` (function) `http://www.kernel.org/doc/man-pages/online/pages/man2/select.2.html`
- `snprintf` (function) `http://www.kernel.org/doc/man-pages/online/pages/man3/sprintf.3.html`

1.2 Functions in tftp.c

The code in tftp.c has 6 functions that you need to extend.

- tftp_connect - The code to establish a connection. Create a socket
- tftp_send_rrq - The code to construct and send a RRQ packet
- tftp_send_wrq - The code to construct and send a WRQ packet
- tftp_send_ack - The code to construct and send a ACK packet
- tftp_send_data - The code to construct and send a DATA packet
- tftp_transfer - The code to manage an entire TFTP session, taking care of sending or receiving the entire file.

The four functions that construct and send packets are very similar, completing one will make the others easier.

1.3 Detailed Program Flow

The tftp client operates in two different modes, depending on if you are getting or putting a file.

Make a Request

1. Open network socket to the server
2. Open a file
3. Construct a read or write request
4. Send request to the server

If Putting a File

1. Read data from file
2. Place data from file in packet to send to server
3. Send data packet to server
4. Receive acknowledgement from server
5. Repeat above steps until data read from file is less than 512 bytes

If Getting a File

1. Receive data packet from server
2. Write data to file
3. Send acknowledgement to server
4. Repeat steps 5 to 7 until data written to file is less than 512 bytes

Finish transfer (Already Implemented)

1. Close file
2. Close network socket

1.4 Planning Your Work

We highly recommend that you plan your work and decide in what order to do things. Here we present a few tips on what might be a good way to start. However feel free to do it anyway you like.

- Implement the functions according to the order in the list in Section 1.2.
- Implement the functionality to GET a file first. It is easier to verify that you solution works since you will have the files on your computer (to check filesize and md5).
- When you know that GET works perfectly implement PUT. Since you do not have access to the server you have to first PUT the file and then do GET on the same file in order to know that it works as it should. Keep in mind that the 3 example files are write protected.

1.5 A Few Helpful Details

- Snprintf takes a string and outputs a result according to which format is specified. Snprintf is preferable to sprintf as it forces the programmer to specify the length of the string being used. It is important to note that the length field of snprint refers to the size of the buffer being manipulated (e.g., 'str' from the manpage), not the length of the string being placed into the buffer. This is because strings are typically null-terminated, i.e., they contain a trailing zero. Since TFTP uses padding to refer to the end of the string, it is not necessary to add a trailing zero yourself. A good sign that you're doing something wrong is when the string 'datakom2' appears as 'datakom' in your packets.
- Do not use malloc(), there is no need if you use the provided stucts. Most people will use malloc() incorrectly, so if you do not know what you are doing, avoid it.
- the TFTP MODE to use is octet
- Hostnetwork byte order: Different computer architectures store the most significant bit of a binary number in different places. There are two types: 'big-endian' (most significant bit first) and 'little-endian' (most significant bit last). In order to avoid confusion since networks consist of different types of communicating computers, 'network byte order' was chosen as a standard. Network byte order is big-endian. In order to cope with this, various helper functions are implemented by most Operating Systems to allow a programmer to easily convert

between host and network byte order. In order for you to send information over the network, it must be converted into the correct byte order.

- Pointers: You cannot send information referenced by pointers over the network. This kind of information must be dereferenced first

- Block number: You will need to keep track of which block you are sending and receiving so that you know how to deal with erroneous block numbers.

- Signed vs. unsigned: binary numbers often have a 'sign' bit which allows the programmer to know if a binary number is positive or negative. Sometimes this information is not necessary (for example, you cannot have a port number with a negative value). In order to maximize the range of number allowed by a particular number of bits (e.g., 16 bits), a binary number can be 'unsigned'. This means that the full 16 bits can be used for expressing a positive number. You will encounter both signed and unsigned values whilst writing your TFTP client.