TFTP Server

Made by: Poonam Mishra (2015B2TS0882P)
Abhinav Gupta (2016A7PS0011P)

Simple TFTP server written in C, based on RFC1350.

Requirements

This server is only unix compatable and requires **glib2.0** to run.

Running

From the project root's directiory, this is an example for port 12345 and access only to a folder called data in the root.

```
$ make -C ./src
$ ./src/udp_select 12345 data
```

To run a client and get a file test file also added in the current project, test.txt, run

```
$ tftp 127.0.0.1 12345 -c get test.txt
```

Design Specifications

This TFTP server doesn't support any WRQ. In case it comes from a client, it'll be responded with an error message.

Features

- Mulitple clients can use the server at once
- Resends in case of block numbers mismatch
- Timeouts for inactive clients, calculated dynamically
- After a certain number of incorrect ACKS, client is removed. This has been set to 3 attempts as per the question
- Timeout of 5 seconds initially to receive ACKs has been set, this is updated dynamically using Jacobson's algorithm.
- Karn's algorithm for RTTI updates is also used
- Only Octet mode is supported
- The server restricts access to the parent directory

Implementation

The server loop runs until interupted from keyboard or an error occurs that results in terminating the process. The loop has three main parts.

- Wait for something in the socket, done with select() and I/O Multiplexing.
- Read from socket, by using recvfrom and a buffer.
- Process buffer that was read into. This is either RRQ, ACK or ERR, all other are not allowed.

Additionally, if nothing is on the socket for a specific amount of time, we go through the client pool and remove those that have been inactive for some time (or after 3 failed attempts).

Starting new transfer

Assuming client does not already exist, we check if this filename contains two dots for parent directory access and if requested file exists. Also we validate the transfer mode and only allow and octet. Failure in any of these will result in an error package sent and the client won't be added to our pool of clients. Otherwise, we read the next 512 bytes from the file and send the first package and also add this client to the client pool (dictionary), a data structure taken from glibc2.0
If he already exists there generally should not be sending more RRQ. In case there is, block number calculation is done and if is still at 1, we resend the first package up to some amount of resends. If they are reached we send an error and remove the client.

Continuing existing transfer

First we check if client exists in our pool. If not, it must not send any ACKs so we respond with a error pack. If it does exist, we check if block numbers match and if not, we resend the last package up to a resend quota, which upon reaching we send an error pack and remove the client from the pool.
If the block numbers do match, we check if the package size was not full which would mark the end of this transfer and the removal of this client from our pool. If we sent a full package we reset the resend counter variable to `0`, increment the block number (if at max value, set to `1`), read the next 512 bytes from file and send it.

Reading from file

If mode is octet we read each byte as is with `fread()`. If not, we had to replace all `\n` and `\r` with `\r\n` and `\r\0` respectively. This is because unix TFTP clients will remove `\r` in netascii mode since they expect windows line feeds to be sent to them. If a binary file is sent, those characters have nothing to do with new lines so the client would be removing bytes essential to the file.