Assignment 3 report

Group effort

Henry 60% Jonathan 40%

## Explanations

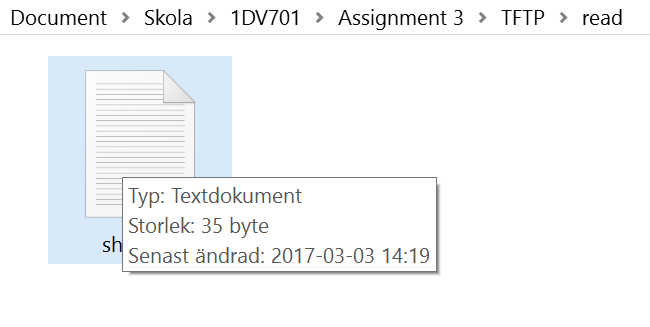
“*GetCanonicalPath ()*” this method [from the File class] will take a path e.g. “root/dir/../a.txt” and calculate the new path which is “root/a.txt”. As this can be a major access issue I use it to check if we are still in the root folder.

Getting bytes to int: You may have seen “((receive [2] << 8) | (receive [3] & 0x00ff))” or something similar. What this does is: getting the value from 2bytes, it starts with the highest byte and moves the bits 8 step to the left (left shift), the second part removes the higher bits. Why use it? Well to make it faster and honestly because it looks more clean then creating a ByteBuffer, allocating space and adding the bytes and convert it (it is enough that I do it when converting int to 2bytes).

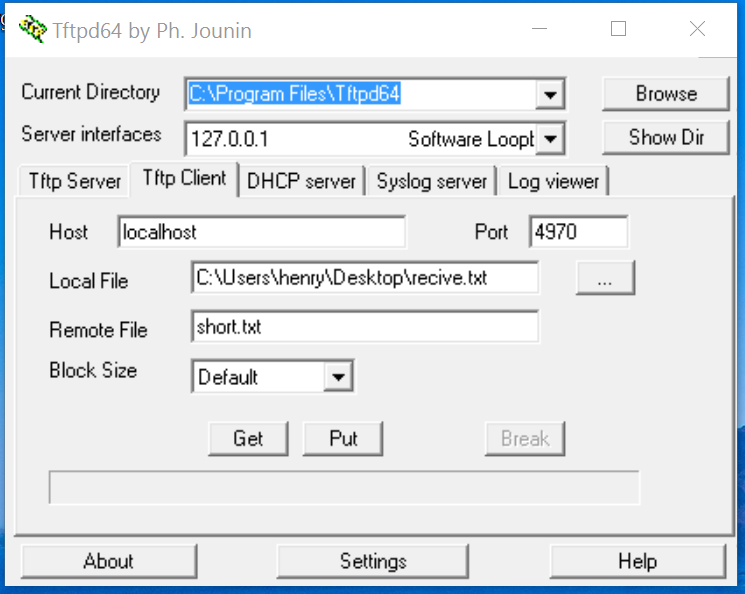
*Arrays.copyOfRange*: is used to create and Array with (value, start, end) parameters. Which in the case of getting the byte array from the datagram is the best way to do so. Since datagram actually is the size of 512 but only read for example 321 bytes, the method getLength of the datagram gives the 321 but the getData gives all (the useless 191 bytes too) so copyOfRange lets you specify the start and the end.

# Problem 1

Receiving a file that is less than 512 bytes.

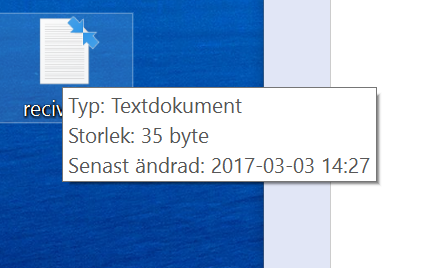


[A file that is 35bytes]



[The TFTP setup]

Now we do a get request and we will check if our empty recive.txt will be 35bytes



[The received file after Get request]

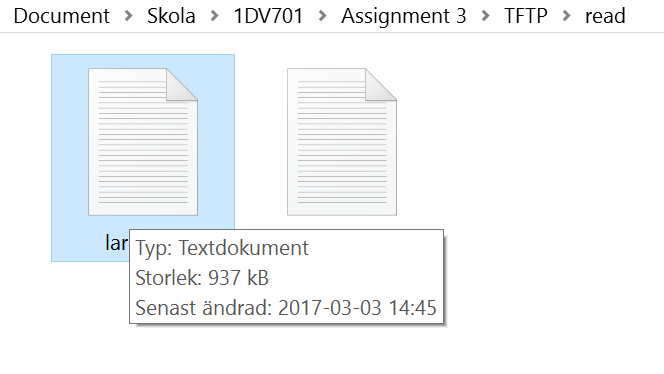
And as we can see, all the 35bytes which is less than 512 was sent.

So why are we using both socket and sendSocket?

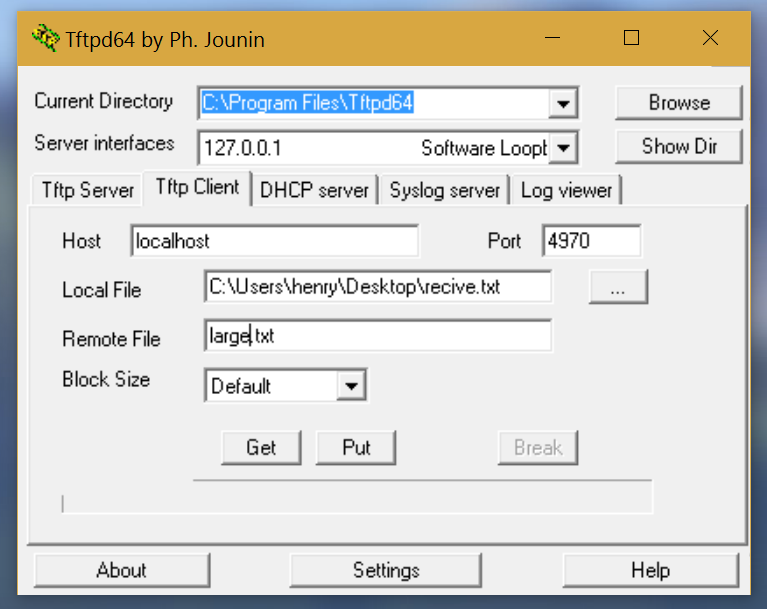
Because we use the socket for listening and when a new client connects we create a new connection for that specific client which we call sendSocket.

## Problem 2

Receiving files that is larger than 512 bytes.

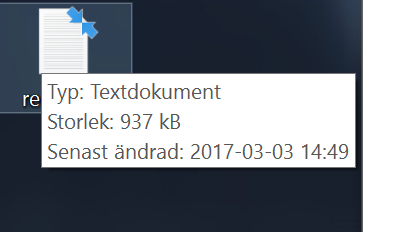


[A file that is larger than 512bytes]



[The TFTP setup]

Now we do another get request and check if our recive.txt will be equivalent to our large.txt file on the remote side.



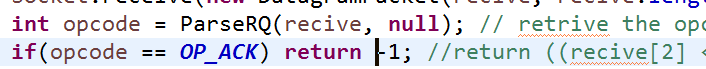
[TFTP Result]

Which it is, now we can send files that is larger, less or equal to 512 bytes.

## Problem 3

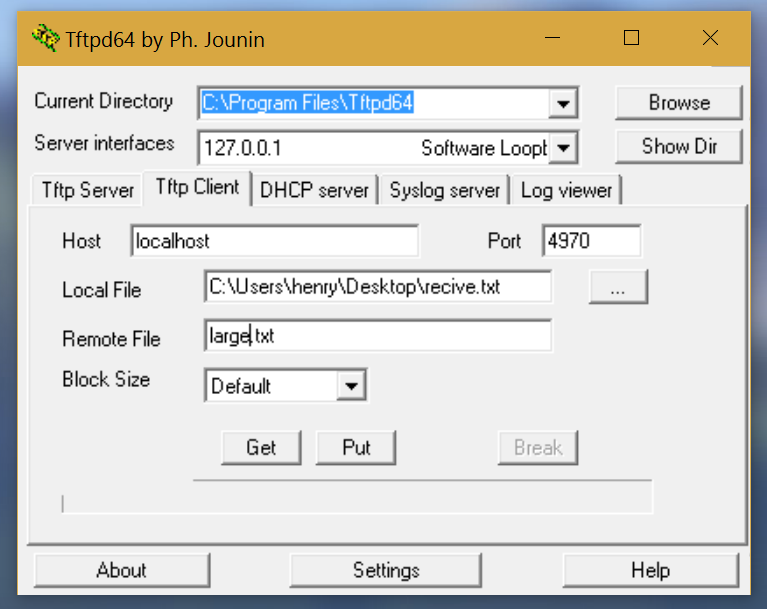
Showing the different errors (0, 1, 2 and 6).

Error 0: Not defined. This happens when retransmissions are too many, which we will fake with a timeout of 0ms.

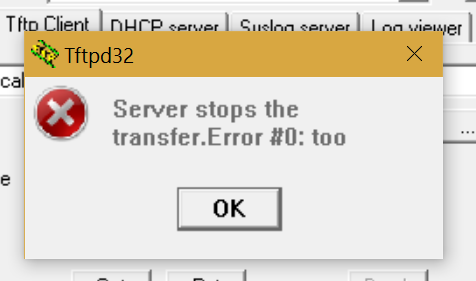


[Faking image]

Requesting the same large.txt file again

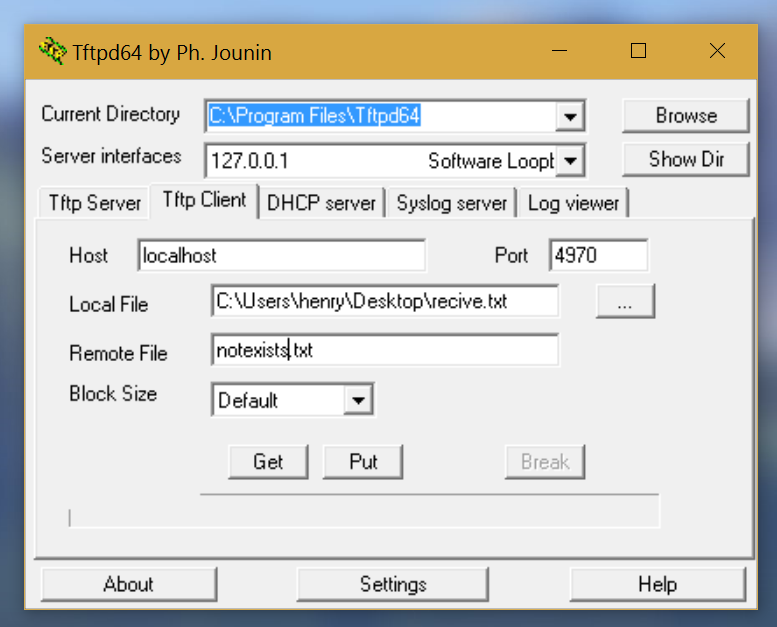


[TFTP setup]

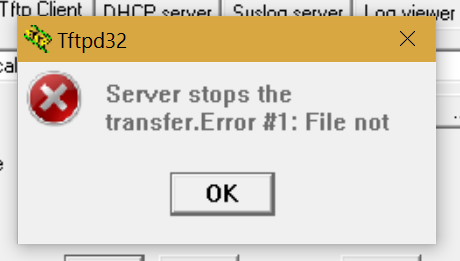


[Result image for error 0]

Error code 1: File not found

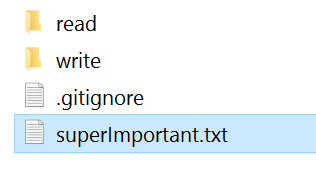


[TFTP setup]

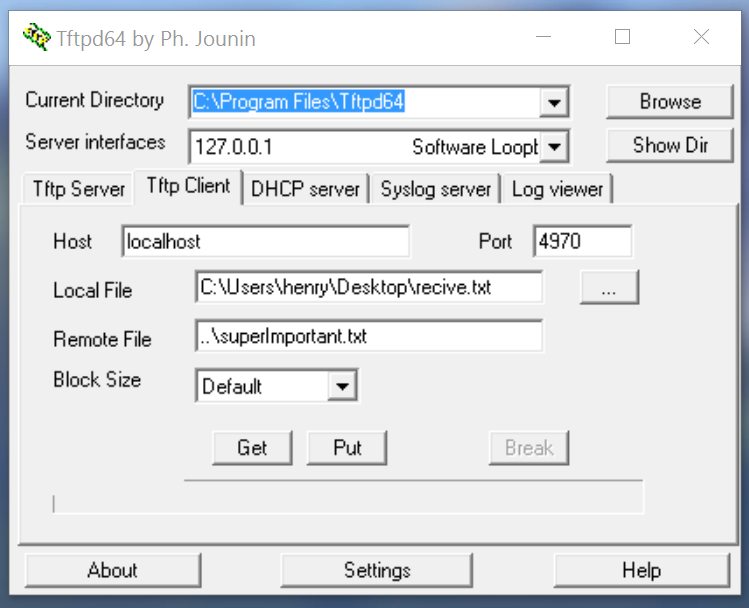


[The result]

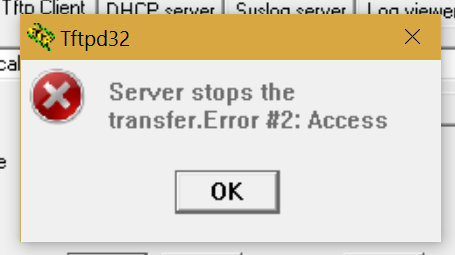
Error code 2: Access violation



[Important file created outside]



[TFTP setup]

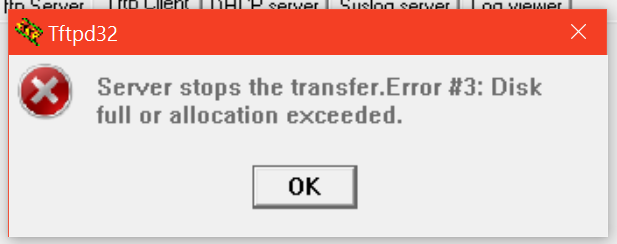


[Result image]

Error code 3: Disk full or allocation exceeded

The file size limit for this server is now set to 800kb

So by sending a file larger then that gives the error 3.

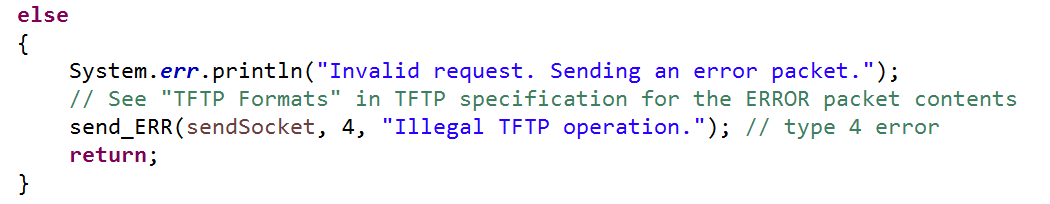


[Result from putting a larger file than 800kb]

Error code 4: Illegal TFTP operation

This error is caused if you make another operation than Write or Read.

If you are using Linux then maybe you can try this, but since we only have windows we can’t.



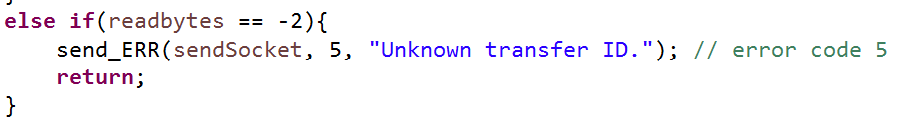
[Code that shows error code 4]

Error code 5: Unknown transfer ID

Caused if the port are not the same when receiving the ACK



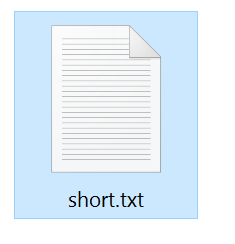
[The port checking]



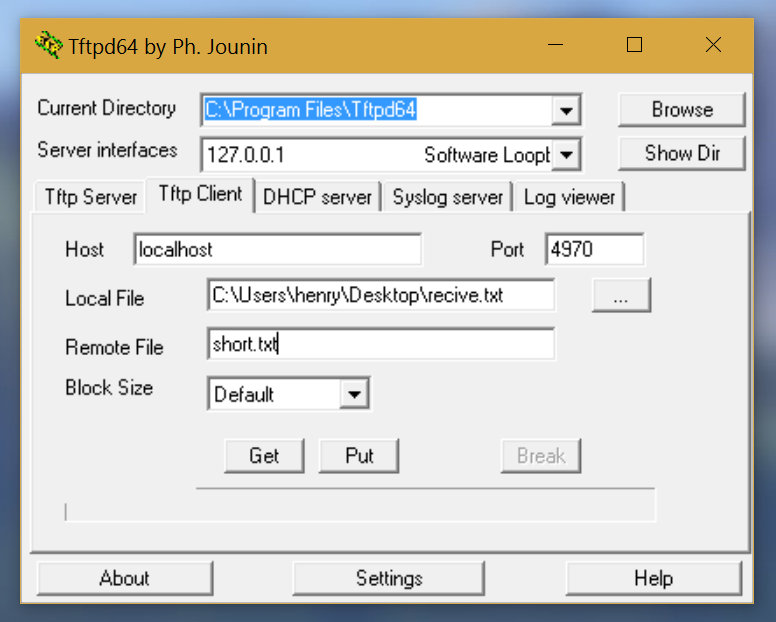
[And the result if this happens]

Error code 6: File already exists

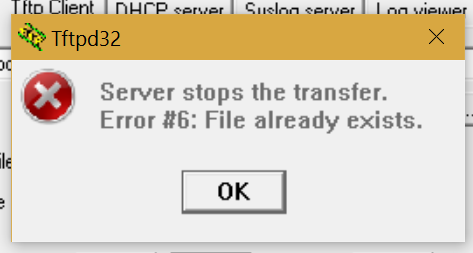
This will be achieved with a PUT request.



[File created in write folder]



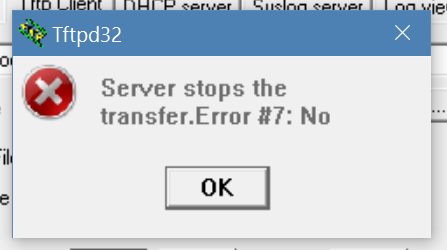
[TFTP setup]



[Result image]

Error code 7: No such user

Will result if you want to access /admin folder which only IP 1.3.3.7 can



[Error 7 result image]