CSCI 351 Programming Assignment 2

Due Date: Monday October 27, 2014 at 11:59:00pm

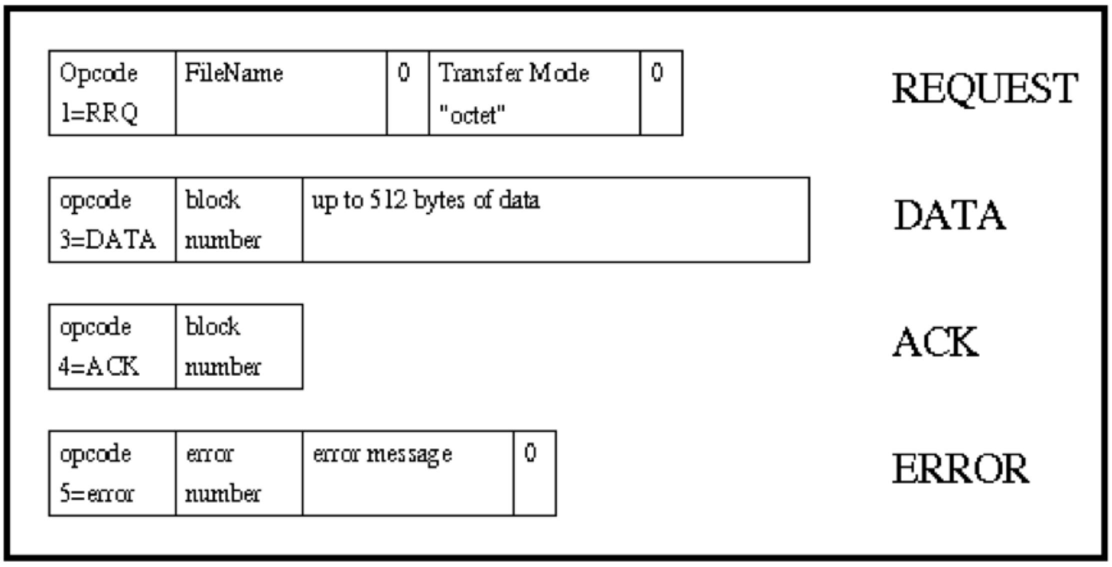
Late projects will not be accepted and will result in a 0

Overview

TFTP is the trivial file transport protocol, which was designed to be used when bootstrapping diskless systems. TFTP was designed to use UDP, to make it simple and small. Implementations of TFTP (UDP, IP, and a device driver) can fit in read-only memory or devices so that they can network boot - this was the initial intent of TFTP. RFC1350 is the official specification of version 2 of TFTP. In this assignment you will write a TFTP client, using Java or c#, that will be capable of downloading files from a standard TFTP server.

The TFTP Protocol

The format of the five types of TFTP messages are illustrated in the diagram below:



Note: Opcodes are 2 bytes long (RRQ would be 01, ACK is 04)

Each exchange between a client and a server begins with the client asking the server to either read or write a file for the client. The read (opcode 1), or write (opcode 2), request includes the name of the file to transfer and the mode in which the transfer should take place. The strings representing the name and the transfer mode consist of ASCII characters terminated by an ASCII zero.

TFTP supports two transfer modes: `netascii` and `octet`. The mode string is either the string `"netascii"` or the string `"octet"` (case is not significant). In a `netascii` transfer,the data consists of lines of ASCII text followed by a carriage return ***and*** a linefeed (CR/LF). The server and the client are responsible for converting between netascii format and the format used by the local host. An `octet` transfer treats the data as 8-bit bytes with no interpretation.

When a server receives a read request, it checks to see if the file exists and determines if it can be read. If the file can be read, the server responds by sending a data packet that contains the first 512 bytes in the file with the block number set to one. When the client sends an acknowledgement for block one, the server will respond by sending a data packet containing the second block in the file. TFTP uses a *stop-and-wait* protocol, which means the server will not send an additional data packet until it receives an acknowledgement for the packet that was just sent. This process continues until the file is transferred. When the client receives a data packet with less than 512 bytes of data, it knows that it has received the final packet (if the number of bytes in the file is a multiple of 512, a data packet of length zero will be sent).

The final type of TFTP message is the error message, which is sent by the server to the client to indicate error conditions. The server will send an error message if a read or write request cannot be processed or if read and write errors during transmission. After an error message is sent the server terminates the transmission. The error number gives a numeric code that identifies the error and the error message is a zero-terminated ASCII string that contains additional information.

Since TFTP uses UDP, it is up to TFTP to handle lost and duplicated packets. Lost packets are handled using a timeout and retransmission policy. Both the opcode and the block number in the five TFTP messages are 16-bit unsigned integer values. Note that the values that you place in these files must be stored in standard network byte order (i.e. Big Endian order) This may not be the byte ordering used on the machine on which your client is running. For further details regarding the TFTP protocol refer to RFC1350.

Requirements

You are to write a program named, `TFTPreader`, that will read a file from a standard TFTP server. The program will take three command line arguments: the transfer mode to use, the name of the host on which the TFTP server is running, and the name of the file to transfer (the TFTP server is located at the well-known port 69). If the program is not invoked correctly (i.e. wrong number of arguments, or an invalid machine name), an appropriate error message will be printed (see sample output below) and the program terminates without generating any additional output.

The program will then connect to the TFTP server and transfer the specified file. If the transfer is successful, a file with the name specified in the command line will be created in the directory in which the program was started. The file will contain the data that was transferred from the TFTP server.

If in the course of the transfer, the server sends an error packet, the program should print a message indicating the error number and error message sent in the error packet. After printing this information the program will terminate without producing any additional output.

A sample run of the program is shown below:

```
kiev> mono TFTPreader.exe
Usage: [mono] TFTPreader.exe [netascii|octet] tftp-host file
kiev> mono TFTPreader.exe netascii glados foobar.txt
TFTPReader: Error code 2: Access violation
kiev> mono TFTPreader.exe netascii glados raven
kiev> ls -l raven
-rw------- 1 jsb fac 7041 Apr 11 15:42 raven
kiev> mono TFTPreader.exe octet glados shuttle.jpg
```

Note the error code/message returned by the TFTP server may differ from what is shown above, and the permissions of the file created by the TFTPReader program may vary as well.

System Information

An implementation of the TFTP server is currently running on the machine glados (glados.cs.rit.edu). This is the ***only*** machine on the CS network, that you have access to that is running the TFTP service. The server is configured so that it will not accept any write requests and will only serve the files that are contained in the directory `/local/sandbox` (you have access to this directory from your CS account when logged onto glados). You will have to change the permissions on the files once you copy them by running `'chmod 755 <filename>'` where <filename> is the name of the file you uploaded.

Note that the Ubuntu TFTP server accepts requests on port 69 but services those requests on a *different* port. This means that your client will send the initial request to transfer a file to port 69 and will receive the packets that make up that file from a different port selected by the server. The client must send the acknowledgements for each packet it receives to the port from which the packet was sent. If you send acknowledgements to port 69 they will be ignored.

You may find it useful to get a tftp program to see how that works. Often times, examing the data that is sent with an existing, functional program can help you understand why you program isn't working. One such tool that can help you diagnose data is wireshark.

Style

Write clean, modular, well-structured, and well-documented code. Here are some general guidelines:

1. Make sure that your name appears somewhere in each source file.
2. Each class and every method in the class should have a header that describes what the class/method does.
3. You do not have to document every single line of code you write. Provide enough documentation so someone who is not familiar with your program can figure it out
4. Names of classes, method, constants, and variables should be indicative of their purpose.
5. All literal constants other than trivial ones (e.g. 0 and 1) should be defined symbolically.
6. Define functions as necessary. Keep your code modular and easy to follow.

Submitting

Place all your source files in a single directory and zip the files up (in zip format). Submit the archive to myCourses, in the project 2 folder.

You may include a README file in this directory, if you wish, but no other files are acceptable.

Improperly submitted projects will results in a minimum of one letter grade deduction.