CSC 4200/CSC5200
Computer Networks, Fall 2018
Programming Assignment 2
Assign Date: October 9, Due October 25 1:30 pm

For this assignment you will implement an in-band ftp protocol. You have to implement an ftp server and an ftp client. You have to use C/C++ and UNIX environment for this assignment. You can make assumptions wherever necessary or ask the instructor/TAs.

The server will run on a particular computer and will listen on a predefined port, which will be passed as a command line argument (use a port number > 2000). It will accept a TCP connection from any client. The client will take two command line arguments: server name and server port number. The client will establish a TCP connection to the server.

The ftp client program should implement at least three user commands: **quit, put** <filename> and **get** <filename>. When the client program is run it should establish a TCP connection to the server and wait for user commands in an infinite loop. The **put** command transfers a file from the local machine to the remote machine. The **get** command transfers a file from the remote machine to the local machine. If the user enters **quit** the client program should close the connection to the server and exit.

The file transfer protocol works using two-part messages between the client and server. A message is a character string terminated by NULL and consists of a command and arguments separated by a ":". The syntax is as follows:

command:<Arguments> No space before or after the colon

Protocol Overview

The client will open a TCP connection to the server. To transfer a file from the local file system to the remote file system the client will send a store (STOR) message along with the file name. When the server receives the message if it wants to receive the file it must send a clear to send (CTS) message. The server may not want to receive the file for reasons such as disk is full, there is already a file with the same name owned by another user, etc. In that case the server will send an error message (ERR) with an error code and description. If the client receives a CTS message it will then send the content (CONT message) of the file in a single message. If the client receives the ERR message it should inform the user appropriately. When the server receives the file content it will create a file in the remote file system and store the content of the file. The name of the file will be the same as the name sent with the STOR message.

To transfer a file from the remote file system to the local file system the client will send a retrieve (RTRV) message along with the file name. The server may not want to send the file for reasons such as file does not exist, file is owned by another user etc. In that case the server will send an error message (ERR) with an error code and description. If the client receives the ERR message it should inform the user appropriately. If the server is willing to send the file then the server should send the content of the file (CONT message) in a single message. The client will create a file in the local file system and

store the content of the file. The name of the file will be the same as the name sent with the RTRV message. The client should be able transfer multiple file to and from the remote file system in a single session.

Protocol Messages

STOR:<filename>

The client requests to transfer the file with name **<filename>** to the remote file system from the local file system. The local file will be unaltered after the action. The client sends this message. The server should never send this message.

Client action: wait for server response

Server action: check whether the file can be received and send the appropriate response message

RTRV:<filename>

The client requests to transfer the file with name **<filename>** from the remote file system to local file system. The remote file will be unaltered after the action. The client sends this message. The server should never send this message.

Client action: wait for server response

Server action: check whether file can be sent and send the appropriate response message

CTS:<filename>

The server is ready to receive the file with name **<filename>** from the client. The server sends this message to client to indicate that client should start to send the file content.

Client action: Send the content of the file using **CONT** message

Server action: Wait to receive the file.

CONT:<ASCII integer>:<file content in ASCII>

ASCII integer – the length of the file content in bytes

file content in ASCII – the bytes of the file

The integer is written in ASCII (as in the chars '0', '1',…)

Server/Client action: write the file content in the file named <filename> and close the file

ERR:XXX <error string>

XXX - 3-digit error code

error string – short error description

error code and error string are separated by a single space

Client/Server action: If client or server receives ERR message they should display the error code and description on stdout.

Define your own error codes and descriptions.

You will need to learn about the following system calls for this assignment

socket(), connect(), bind(), listen(), accept(), send(), recv(), close(), gethostbyname() and few other related function calls. Use man page to learn about these system calls.

Note:

You should use the **pingpong.h** and **pinpong.cpp** that you have developed for assignment one. The server code should be in server.cpp/server.c, and client code should be in client.cpp/client.c. You may assume that files will be transferred to/from the current working directory and we will be dealing with ASCII files. Your client and server will be tested against the server and client written by TAs.

Submission

Submit your program on ilearn (source and header no executable) along with a README file (plain text format, no doc, odt, pdf etc). Put all your source and header files in a folder name lastname_firstname and zip the folder (use gzip, no winzip). Submit the zipped file on ilearn.
Your read me file should clearly describe how to compile and run your program. List your error code and description in the README file.

Grading

Readme	**10%**
Following submission direction	**10%**
Program works properly	**80%**