CSC 4200/CSC5200
Computer Networks, Spring 2016
Programming Assignment 2
Assign Date: February 22, Due February 29 5:00 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.

The server will run on particular computer and will listen on a predefined port which will be passed as command line argument (use a port number > 10000). It will accept 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 wait for user command in an infinite loop. The put command transfers a file from local machine to remote machine. The get command transfers a file from remote machine to local machine. If 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. Each message consists of a command and optional argument separated by ":". The syntax is as follows:

command:[Arguments]

Protocol Overview

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 receive 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. When the server receives the message if it wants to send the file it must send a ready to send (RTS) message. The server may not want to send the file for reasons such as file does not exists, 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 receive the ERR message it should inform the user appropriately. If the server send and RTS message in response to client's RTRV message the server should send the content of the file content of the file (CONT message) in a

single message. The client t 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>

Transfer the file with name **<filename>** to the remote file system. The local file will be unaltered after the action. Client sends this message. Server should never send this message.

Client action: wait for server response

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

RTRV: <filename>

Transfer the file with name **<filename>** from the remote file system to local file system. The remote file will be unaltered after the action. Client sends this message. Server should never send this message.

Client action: wait for server response

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

CTS: <filename>

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

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

Server action: wait to receive the file. Once the file is received create a file in the local file system (server's file system) and store the content.

RTS: <filename>

Server is ready to send the file with name **<filename>** from the server. Server sends this message to client to indicate that server is about to send the file content.

Client action: Wait to receive the file. Once the file is received create a file in the local file system (client's file system) and store the content.

Server action: Send the content of the file using **CONT** message.

ERR:XXX <error string>

XXX - 3 digit error code

error string – short error description

error code and error string is separated by a single space

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

Define your own error code and description

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.

Submission

Submit you program on ilearn (source and header no executable) along with a README file (plain text format, no doc, odt, pdf etc). You should a have single source file for server and single source file for client. The name of your source files should be as follows:

lastname_firsname_Server/Client.cpp/c for example my client source file name would be

ghafoor_sheikh_Client.c or **ghafoor_sheikh_Client.cpp**

Your read me file should clearly describe how to compile and run your program clearly. The read me file also should clearly explain the user commands that your client implemented. List your error code and description in the README file.

Grading

Readme	**20%**
Program works properly	**80%**