Socket Programming Exercise: SimpleFTP

This socket programming project is inspired by those of you who used "email" to transfer a file from a VM guest machine to the VM host.

1. In this exercise, you have to write a client and server program, which uses TCP sockets underneath, to transfer a file from the server to the client.
2. The program can be written in phases, with increasing levels of functionality in each step.
3. The exercise is to be done individually.

Phase-1: Plain File Transfer [6 marks]

1. Server called SimpleFTPServerPhase1.cpp
    a) 2 command line arguments: portNum, fileToTransfer
        i. Print usage on stderr and exit with exit-code 1, if wrong number of command line arguments are given
        ii. Print appropriate error message on stderr and exit with exit-code 2, if bind on given port fails
        iii. Print appropriate error message on stderr and exit with exit-code 3, if given file not present or not readable
    b) Server should listen on given port for incoming connection from client
        i. STDOUT on successful bind "BindDone: portNum\n"
        ii. STDOUT on successful listen "ListenDone: portNum\n"
        iii. STDOUT on incoming connection "Client: ipaddr:port\n" where ipaddr and port are the client-side info
    c) Transfer the file over the TCP connection
        i. STDOUT on successful transfer "TransferDone: **??xyz bytes**\n" where xyz is the size of the given file
    d) Server can exit after one successful file transfer
2. Client called SimpleFTPClientPhase1.cpp

a) 2 command line arguments: serverIPAddr:port, fileToReceive

   i. Print usage on stderr and exit with exit-code 1, if wrong number of command line arguments are given

   ii. Print appropriate error message on stderr and exit with exit-code 2, if connection to server fails

   iii. Print appropriate error message on stderr and exit with exit-code 3, if unable to create/write the given/received file

b) Form a TCP connection to the given serverIPaddr:port

   i. STDOUT on successful connection "ConnectDone: serverIPAddr:port\n"

c) Receive the file from the socket and write to local file name as given in command line argument

   i. STDOUT on successful file reception "FileWritten: **??xyz bytes**\n"

d) Client can exit after one successful file reception

3. Marking scheme:

   a) Correct exit codes at server: 1

   b) Correct exit codes at client: 1

   c) Correct file transfer: 4

Phase-2: File Name from Client [3+1=4 marks]

1. Server called SimpleFTPServerPhase2.cpp, similar to phase1, except for the following

   a) Only one command-line argument, the portNum

   b) Until "incoming connection from client", behaviour is same as phase-1, including various STDOUT

   c) After forming incoming connection from client, read a null-terminated string of the format "get fileName" where fileName is the file requested by the client

      i. STDOUT after this "FileRequested: fileName\n"

      ii. If incorrect format of string from client, STDOUT "UnknownCmd\n", also print appropriate error message on

STDERR, and close the client connection

iii. If file not present or not readable, STDOUT “FileTransferFail\n”, also print appropriate error message on STDERR, and close the client connection

d) After this, file transfer behaviour similar to phase-1, STDOUT after transfer similar too

e) DO NOT exit after file transfer, instead wait for the next connection from another client (i.e. server never exits, is in a loop)

2. Client called SimpleFTPClientPhase2.cpp, similar to phase1, including command line arguments

a) On successful server connection, send null terminated string “get fileName” for the file get request over the TCP connection

b) Remaining behaviour same as phase1 client

3. Marking scheme:

a) Correct behaviour for fileName from client: 3

b) Server supports many clients, one after another: 1

Phase-3: Simultaneous Multiple Clients [3+2=5 marks]

1. Server called SimpleFTPServerPhase3.cpp, similar to phase2, except that it should support at least 2 simultaneous clients

a) Note that the different simultaneous clients could have asked for different files

b) You *do not* have to use multiple threads (or multiple processes) for this; in fact I recommend strongly that you instead use the select system call appropriately

2. Client called SimpleFTPClientPhase3.cpp, similar to phase2, except the following

a) It should take a third command-line argument receiveInterval

i. The client should receive a maximum of 1000 bytes every receiveInternal (in ms)

b) One of the purposes of this rate limiting feature of the client is to be able to easily test multiple simultaneous clients at the server

c) Note that no rate limiting is required at the server side; TCP takes care of

the flow control!

3. Marking:
    a) Correct server implementation: 3
    b) Correct client implementation: 2

Phase-4: File Transfer in Both Directions [2+2+1=5 marks]

1. Server called SimpleFTPServerPhase4.cpp, similar to phase2/phase3, except the following
    a) On successful incoming connection, read from the client either "get fileName" or "put fileName"
    b) If its a put command, the server should consider everything after the "null" of the above string as contents of the file to be received, and write to that file locally
2. Client called SimpleFTPClientPhase4.cpp, similar to phase2/phase3, except to support the put command as well
    a) Command line arguments now: serverIPAddr:port, op, fileName, receiveInterval
    b) Here op is either get or put
3. Marking scheme:
    a) Correct server: 2
    b) Correct client: 2
    c) Server still supports simultaneous clients: 1