Concordia University	**COMP 445 - Fall 2014** **Laboratory Assignment 1**

Due Date: September 26, 2014 at 11:59 PM

File Transfer Protocol

Specifications and requirements

The goal of this first laboratory assignment is to acquaint you with the basics of network programming within the TCP/IP Internet family environment, and its socket application programming interface (API) and also a little bit of multi-threading. You will use the TCP protocol as a "delivery service", to carry your packets. The particular problem that you have to resolve consists of transferring files between multiple clients and one server logged in at different computers.

You must write your program in C/C++, and you must be prepared to demonstrate your program in the XP environment in the lab. You may develop your programs elsewhere, but the demonstration must take place in the XP lab. In particular, Java and/or UNIX versions of the programs are NOT acceptable.

The basic scenario unfolds as follows:

1) AT THE SERVER: one partner logs at one computer [say sun], and starts one application program to act as the file transfer server (call it ftpd_tcp.cpp). At startup, program ftpd_tcp displays the following messages:

→ ftpd_tcp starting at host: [sun]
waiting to be contacted for transferring files...

2) AT THE CLIENTS:

2.1) the other partner logs into the second computer [say earth], and starts the other application program to act as the first client (call it ftp_tcp.cpp). At start up, program ftp_tcp displays the following messages:

→ ftp_tcp starting on host: earth
Type name of ftp server: sun
Type name of file to be transferred: fristfile.xxx
Type direction of transfer: get

Sent request to sun, waiting...

2.2) Your partner logs into the third computer [say mars], which acts as the second client, and starts the client (ftp_tcp.cpp). Similarly, client displays the following message at startup:

→ ftp_tcp starting on host: mars
Type name of ftp server: sun
Type name of file to be transferred: secondfile.xxx
Type direction of transfer: get

Sent request to sun, waiting...

3) AT THE SERVER: the server receives the request, displays the identification of the requestor, the name of the file to be transferred and the direction of transfer:

User "spacewalker" requested file somefile.xx to be sent.
Sending file to earth , waiting...

User "spacewalker" requested file somefile.xx to be sent.
Sending file to mars, waiting...

4) The server puts out a message stating that the file has been completely transferred, and returns to the waiting state.

5) AT THE CLIENT:

The client puts out a message noting the arrival of the first bytes of the file, and another message noting the completion of the transfer. It then returns to ask for another transfer. Entering an ftp servername of "quit" should cause it to exit. Error messages should be displayed at the client if you attempt to put or get a non-existent file. It should also be possible to print out a list of the files available for transfer on the server on the client machine using a list command instead of put or get.

6) AT THE SERVER:

You will have to enter control-C to kill the ftpd_tcp program.

To develop these two application programs, you should consult several sample programs provided to you (A: a sample server and client cli_tcp.cpp and ser_tcp.cpp. B: a C++ thread class implementation Thread.cpp and Thread.h), and modify them to suit the requirements of the present lab assignment. An acceptable value for the direction is "get" or "put". "put" causes a file to be copied from the client to the server.

Notes:
1. You will have to design a simple protocol to be used between the client and the server,

to identify an incoming packet as a request or a response or user data. Clearly the file name, the name of the user, and the direction of transfer have to be sent from the client to the server using this "protocol". You are to obtain the name of the user from the operating system using a system library routine. You must negotiate the initial information with the file server *prior to sending any data*, using your "control packet" format. Your exchange must allow for the possibility of errors at the file server (e.g., file not found, etc.) and report any error messages to the client.

2. Once the initial negotiation is complete, you are to exchange the data. You can use one of two ways to indicate file length: a) exchange file length as part of the initial negotiation, or b) indicate end-of-transfer explicitly in your "data packet" format.

3. TCP is being used here as a "delivery service" to carry the "packets" of your "simple protocol". The virtue of TCP is that it will never lose any of these packets, and it will not mis-order them. In the subsequent laboratories, you will use a "delivery service" that explicitly loses packets (randomly), and may delay packets. Try to think about how this will affect the operation of your "simple protocol", and "plan ahead" for these labs.

4. In order to add the multi-threading feature, you are supposed to use the _beginthread provided by Windows Operating System, which is the counterpart of the pthread_create in Unix environment. If you are using C++ for your program, one C++ class, which is wrapping the _beginthread, has been provided to you (Thread.cpp, Thread.h). You can just define your own class which inherits from the Thread and override the run() function. Please see section 5.3 for a sample program. If you are using C for your program, please see the section 5.2 for a sample program.

5. Several supplementary sample programs, which you can use for your programming assignment1.

5.1: A server/client without multi-thread: no-multithread.zip

5.2: A server/client with multi-thread in C language: multithreadC.zip

5.3: A server/client with multi-thread in C++ language: multithreadCPP.zip

Groups

For this lab assignment (and for the ones to come), a team of two is permitted, and no bonus is given for working alone. However, if it is discovered that groups larger than 2 have formed, the penalty will be a mark of zero (0) for *all members* of such larger teams.

Deliverables (See course outline for submission format. More details are also available at the course website. You need to look at these details/instructions prior to submitting your assignment)

1. You are required to implement the file transfer in *both direction*s, from the Client to the Server, and from the Server to the Client. The protocol that you use to establish the transfer must accept both "get" and "put" as directions. You must also show the ability to do the list command and display appropriate error messages.
2. Submit your assignment online (at ***https://fis.encs.concordia.ca/eas***) by the deadline. Following that, there will be a demonstration with the lab instructor to go through your work. The demonstration will be performed using the submitted version. The lab instructor will require that you reserve a time for your demonstration. He will announce the procedure through the mailing list of the course, as appropriate. It is your responsibility to reserve your demonstration time enough in advance that you will be ensured of being able to demonstrate your work. The lab demonstrators are permitted to refuse requests once their slots are full.
3. At the time of demonstration, the lab instructor will ask you questions about the functioning of the program; any student of the group must be able to answer any question. Part of the marks will be assigned for demonstrating compliance with requirements at this demonstration. Marks assigned to each member of the group may be different, depending on ability to answer the questions.

NOTES:

1. The first time you use the online submission site from a particular computer, you may be asked to accept a "locally-generated" security certificate. As long as the certificate refers to eas.encs.concordia.ca, please accept it permanently.
2. **PLEASE do NOT make use of any "program generator" for this assignment.** The purpose of the assignment is to ensure that you know how to write network programs using the socket interface, not to demonstrate skills in fancy C++ programming. You may hence write your code in basic C or C++ styles that would run in a plain Windows or Unix environment.