Homework 4

UDP File Transfer

CSCI3550 (Fall 2022)   Due: Fri/Oct/21/2022 at 11:59 AM (Noon)

(100 points) In this assignment you will develop a "simplified" version of FTP to transfer a text file (source file) from the client to the server using UDP. The client reads from the source file, one line at a time and sends the line to the server. The server updates the destination file by writing/appending each line received from the client.

Indeed, this assignment builds on the client/server UDP program code discussed in class. Recall in that program the client sends a short string of characters to the server, which is then sent back to the client and displayed by the client. In this assignment, you will transfer a file to the server rather than sending a string. You will save time by working on the program solution provided to you in the slides.

- The client program accepts three arguments from the command line. Specifically, to run the client program, the command line should look like: *client-program server-name server-port source-filename*. See below for an example.
- On the server side, the command line has two arguments: the port number and the target (destination) file. The port number is obviously the server port number, and the target file is the name of the file that is going to receive (written into) the file. Therefore, the command line looks like: *server-program server-port destination-filename*. See the following for an example. Note that the server-port on the client and server command lines are the same. The server runs in a loop and able to receive different text files but able to write to the same destination file. Each time a file is received the destination file is re-written (no appending).
- The server side port number is 52xxx, where xxx is the last three digits of your choice. The xxx could be any 3-digit number, but to avoid conflicting port numbers with your classmates, you can use options such as the three digit of your car license number; or anything else that reduces conflicts as much as possible.

To hand in the assignment, zip your files in *odin.unomaha.edu* and submit via Canvas. You should be able to zip the files in Windows as well, but try it in *odin* to be sure I will be able to unsip the files when testing your program in *odin*. To zip your files, use the following command in the current directory: *zip your-name file1 file2 file3*. This will create a zip file called *your-name.zip*. Then submit your-name.zip via Canvas. For example, if I were to submit the assignment, I would do:

➢ zip azad ftpclient.c ftpserver.c headerFile.h

Then I would submit azad.zip.

Continued on next page →

This is how I test your programs for grading on *odin.ist.unomaha.edu*:

- I will check the code and its correctness.
- I will compile and run your program using the commands similar to the following using two windows, one to pretend as a client machine and the other to pretend as a server machine. To test your program, I might also run the client program multiple times (using different text files) but run the server only once until I decide to exit the server program.
    - On the client machine:
        - gcc ftpclient.c –o clientout
        - ./clientout odin 52007 SourceFileName

          File is sent successfully

          Waiting for confirmation…

          File received successfully
    - On the server machine:
        - gcc ftpserver.c –o serverout
        - ./serverout 52007 DestinationFileName

          Waiting to receive a file…

          Received the file and successfully transferred to DestinationFilename

          Do you wish to exit the program [yes][no]?

Remarks:

- It is highly recommended to type, debug, and run the UDP program discussed in class. This will save you a lot of time when you decide to modify the program to transfer a file. Note that the UDP program discussed contains some syntax errors that need to be corrected.
- You can develop and test your program on any system of your choice, but it must compile and run on *odin* correctly. Odin (odin.unomaha.edu) is a Linux server often used for programming assignments.
- Please do not use the port 52007. This is reserved for testing and grading your programs.
- Note that you might start and end the client multiple times, whereas the server keeps running until exited.
- I will provide you with the text file that you can use as the source file.
- There is already a user account created for you on Odin. Your userid is the same as your university email. For example, my email address is azad@unomaha.edu, and so my userid on odin is azad.
- If you cannot access Odin (e.g., if accessed off campus), you will probably need to install the university VPN. Access the following link for guidelines: https://www.unomaha.edu/news/vpn-updates.php
- To connect/login to Odin, you will need a utility such as PuTTy or MobaXterm. If you need to transfer files between Windows and Odin, you can use WinScp. PuTTY and WinScp along with other useful packages can be installed from http://ninite.com
- You can find more information about Odin and the utilities needed using the following link: https://www.unomaha.edu/college-of-information-science-and-technology/about/odin.php
- To refresh your mind, you can easily find the basic Linux commands online.
- The following pages show the general algorithms for the client and the server for developing your programs. Your code does not have to follow the algorithms provided but your code should include the necessary input/output messages/parameters as stated in the algorithms and use the command line parameters indicated above in order to invoke the client/server programs.

```
// Client program using UDP

Include Headerfile
Define Constants

// Function to send the file to the server
//SourceFileName: File to send to server,
// SocketDescriptor: Socket descriptor of client
//ServerAddress: Socket address of server (or build as global address)
SendFile (SourceFileName, SocketDescriptor, ServerAddress)
{
        Declare variables
        Open SourceFileName     //Open the source file for reading with proper error handling
        Loop until EOF          //Loop until end of source file is reached
        {
                Read Line       //Read a line from the source file
                Send Line       //Send the line to the server using sendto()
        }
        Send EOF                //Send "EOF" to the server using sendto()
}

MainProgram (Argc, Argv)
{
        Declare variables
        Read Argc & Argv        //Read and check correctness of Argc and Argv from command line
                                // and assign to proper variables
        Create Socket           // Create the client socket with proper error handling
        Create Sever Socket     // Build server address and port using server name
        SendFile (SourceFileName, SocketDescriptor, ServerAddress)
        Print (File sent, waiting for confirmation)     //Print a message and wait for confirmation message
                                                        // from the server

        Receive Confirmation    //Use recvfrom() to receive a confirmation message from the server
                                //Use proper error handling if recvfrom() fails
        Close Socket

} // End of client program using UDP
```

Continued on next page →

```
// Server program using UDP

Include Headerfile
Define Constants
Declare global variables (e.g., client address)

// Function to receive the source file from the client and write the file to the destination file
ReceiveFile (DestinationFileName, SocketDescriptor)
{
        Declare variables
        Open DestinationFileName        //Open the destination file for writing with proper error handling
        Loop until EOF                  //Loop until end of source file marker received from the client
        {
                Receive Line            //Receive a line from the client using recvfrom()
                Write Line              //Write the line received to the destination file
        }
        Close DestinationFileName       //Close the destination file
}

MainProgram (Argc, Argv)
{
        Declare variables
        Read Argc & Argv        //Read and check correctness of Argc and Argv from command line
                                // and assign to proper variables
        Create Socket           // Create and bind server socket with proper error handling
        ReceiveFile (SourceFileName, SocketDescriptor)

        Loop until Response = yes       //Loop until the user decides to terminate the server
        {
                Print(Waiting for a file)       //Print a message signaling server is waiting to receive a file
                ReceiveFile(DestinationFileName, SocketDescriptor)

                Send Confirmation       //Send a confirmation message to client using sendto()
                Print(Do you wish to exit program [yes][no])
                Read Response           // Read response from the user
        }

        Close Socket

} // End of server program using UDP
```