CSE 5462: Lab 2 (Total: 100 points)

Demo in CL 112: Sep 9 (Friday)
Electronic Code Submit Deadline: 9pm, Sep 9 (Friday)

Write a file transfer application using TCP sockets. The file-transfer protocol will include a server called *ftps* and a client called *ftpc*. You will first start the server using the command

ftps <local-port>

You will then start *ftpc* with the command

ftpc <remote-IP> <remote-port> <local-file-to-transfer>

The *ftpc* client will send all the bytes of that local file. The *ftps* server should receive the file and then store it. Make sure that the new file created by *ftps* has the same name but it is in a sub-directory called "recvd" to avoid over-writing the original file since all the CSE machines have your root directory mounted. In case the sub-directory does not exist, your program must create it.

The file-transfer application will use a simple protocol. The first 4 bytes (in network byte order) will contain the number of bytes in the file to follow. Use functions like htonl, ntohl, htons, and ntohs to achieve appropriate byte order conversion. The next 20 bytes will contain the name of the file. The rest of the bytes in the TCP stream to follow will contain the data in the file. Note that TCP is not packet oriented, but it is stream oriented. Before interpreting the fields in the application layer header (4 bytes and 20 bytes), make sure that you have read enough bytes from the TCP stream. If not then go back in a loop and read more bytes. Remember that the argument in recv() only specifies the maximum number of bytes to read.

You can use the client and server programs available on the project page to develop the above programs.

Submit **well-documented and well indented** code with a README file and a Makefile using the following command:

submit c5462aa lab2 <code-directory-name>

Note that buffers, arrays or any data structures in any process cannot be more than 1000 bytes.

Note that your program should work for a binary file (images etc.) of any arbitrary size and the client and server should be running on different machines on **stdlinux (e.g., epsilon, zeta, beta, gamma)**.

You can use the sample images on the project webpage to test your program.

Your program must be able to transfer binary files. **Do not use ASCII** functions (e.g., fgets, fputs, strcpy, scanf, sscanf) to manipulate binary data. You may need to allocate enough time (often quite difficult to estimate) for debugging. Follow the exact specifications and requirements as outlined in this document.

Your program does not need to support transfer of multiple files. It is ok if it quits after transferring one file.

The code must be well documented. And there should be a README file explaining the purpose of each file containing any C code in the directory. The README file should clearly list all the steps for compiling and testing your program. The grading rubric is as follows:

- Design: 50
  ftpc: 25
  ftps: 25
- Program correctness and robustness (should work for any text file): 5
- Program correctness and robustness (should work for any binary file): 25
- Coding style (Makefile, indentation): 10
- Documentation (README file, comments): 10

FAQ

Q: How do I log in to two different machines?
A: When you log in to a stdlinux machine or open a terminal you are assigned to an available stdlinux machine from a pool of machines. From there you can do say, "ssh zeta" to ssh to a machine zeta.

Q: How do I find the machine name or IP address of the machine I am logged in?
A: **hostname** and **hostname -i**. Note that ""0" or "0.0.0.0" or "127.0.0.1" are synonymous to my own IP address. You can use "getifaddrs()" within your program.

Q: Is there any difference between read() and recv() when used to receive data from sockets? Or, read() is equivalent to recv() for TCP sockets and is equivalent to recvfrom() for UDP sockets? Similar questions for write() vs. send()/sendto().
A: recv/send allows you to specify certain options (in the flags argument) which are not available with read/write. read/write are more generic for reading/writing from/to many different types of devices such as USB ports. So if you do not use the flags then these functions are similar.

Q: How do I find out if I have enough space?
A: Use "df -h". In addition, you can use

```
du -a ~/ | sort -n -r | head -n 10
```

to figure out what to delete.