UNIVERSITY OF DELAWARE
DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

CISC 450 / CPEG 419: Computer Networks I

Fall Semester, 2019 Professor: Adarsh Sethi

Programming Project 1
Some Helpful Information

1. All programming for this course should be done on the host *cisc450.cis.udel.edu*. You can access this host if you have an *eecis* ACAD account. If you don't have such an account, fill out a request at *https://accounts.eecis.udel.edu*. If you already have an *eecis* account but it is expired, then you should apply at the same site to get your account renewed. Finally, if you have an account but have forgotten the password, there is a link at the site to request a password reset.

2. You can log in to the *cisc450* VM by opening a remote terminal window to it from your laptop. If you don't have an app that gives you remote terminal access, you can download *PuTTY* from http://udeploy.udel.edu. However, this only works if you are on-campus. If you are not on campus, you can only login to *cisc450* through a VPN. Instructions for setting up a VPN on your computer can be found at http://www1.udel.edu/it/help/connecting/vpn/index.html. An easier work-around without using a VPN is to first use PuTTY to login to *go.eecis.udel.edu* from off-campus. Once you are logged in to this host, simply use the command "*ssh cisc450*" to log in to the *cisc450* VM. This command assumes you have the same log in names and passwords on both machines. If that is not the case, you must use "*ssh username@cisc450*" where username is your user name on *cisc450* and you will be prompted to type in your password.

3. The client and server programs discussed in the class are available on the *cisc450* host in the directory */usa/sethi/networks/proj1*. You should copy the files in this directory to your own project directory and use these programs as building blocks for your own code.

4. You should change the port number SERV_TCP_PORT in the server program to a different port number chosen by you so you don't have a conflict with the port numbers being used by the other students in the class.

5. The socket API functions used in the TCP client and server programs (e.g., *send, recv, socket, connect*, etc.) have Unix *man* pages that provide more detailed information on how to use these functions. These *man* pages are not loaded on our *cisc450* VM, but they are widely available on the Web. You can view them by typing, e.g. *Unix man send* in your browser search box.

6. You should know clearly the sizes of the different integer types on the *cisc450* VM, and use them appropriately. An *int* is the same as a *long int* and they are both 4 bytes long, while a *short int* is 2 bytes long.

7. The functions *htons*, *htonl*, *ntohs*, and *ntohl*, should be used on all short/long integers before putting them in the buffer for transmission, and after getting them out of the buffer when

received. Strings and characters do not need such conversions. The conversion must be done individually on each integer; it cannot be done on a *struct* or other compound data structure.

8. Both the *send* and *recv* functions return a value, which is the number of bytes read or written, or a number that is 0 or less than 0 if there is an error. My sample code in *tcpclient.c* and *tcpserver.c* does not use or check these returned values, **but you should be doing so.** The number of bytes sent or received, which are to be printed in the statistics, must be measured from the values returned by *send* and *recv*.

9. The project specification explicitly states that the data read from one line of the input file, including the newline character, should be transmitted in a single data packet. Different lines of the input file may have different number of characters, so different data packets may have different sizes. It is not permissible to merge multiple lines of the input file into a single packet.

10. You should read one line from the input file and transmit it before reading the next line. It is not permissible to read in the entire input file into one giant array or other structure and then transmit the contents from there.

11. The message to be transmitted by *send* does not necessarily have to be a string (or array of char). For example:

```
int i, j, bytes_sent;
scanf("%d", &i); // reads in a value for i
j = htonl(i);
bytes_sent = send(sock_client, &j, sizeof(j), 0);
```

    The above will send out a 4-byte integer on the socket *sock_client*. The same is true for the *recv* function as well. You can also directly send and receive a *struct*, in which case simply use the address of the *struct* as the second argument to the *send* and *recv* functions.

12. For this project, each packet (as described in the project specification) should be transmitted as two separate messages using two invocations of the *send* command. The first *send* should be used to send the packet header and the second *send* should be used to send the data bytes. On the receiving side, two separate invocations of the *recv* command should be used to receive the packet header and the data bytes. The same applies when sending and receiving the filename.

13. You may find it convenient to use a *struct* to send and receive the packet header and an array of characters to send and receive the data bytes. The packet header is always of fixed size. However, the number of data bytes will vary in each packet. On the receiving side, since you have already received the packet header first, you will know the size of the data message that will arrive next. You should specify this size as the third argument of the *recv* function call instead of the maximum size of the buffer being used.

14. By default, strings in C are null-terminated. You need to be aware of whether or not the strings you are using (if any) are null-terminated or not, and whether or not the null character (if any) is being transmitted in the packet. Note that the null character is not the same as the newline (end-

of-line) character. This project explicitly requires that the newline character must be transmitted at the end of each line in a data packet sent by the server to the client and that no null character should be transmitted in these data packets. Also, the filename sent by the client to the server should not include the newline character. Some of you may wish to use the C string library functions to read and write the data from the input file into your packets and from the packets to the output file. Use of these functions is not necessarily wrong, but you must be aware of what these functions do and be careful that *the null character is not transmitted in a data packet or written into the output file.*

15. If you kill your server with a Control-C, or otherwise have an abnormal termination which does not close the server socket, and then try to immediately run the server again, you may sometimes see an error message: “can’t bind to local address: Address already in use”. In this case, you may have to wait for a few minutes before the port is released and becomes available for use again. You may also get this error if someone else is using the same port. If that is the reason for your error, then change your server port number.