CS260 Assignment #2:
Cross-Platform Sockets and TCP

Task

This assignment introduces TCP sockets and transitions you from Windows to Linux.

The structure of this assignment is largely identical to Assignment #1, except that you will be connecting over TCP rather than UDP. That means you will need to go through these steps:

- Initialize
- Create a TCP socket
- Create the remote address
- Connect to the remote server
- Send data over TCP
- Show that you have nothing left to send by calling shutdown()
- Listen for data with recv() until it returns 0
- Close the socket
- Shut down

As with Assignment #1, the remote IP address is 104.131.138.5, and the remote port number is 8888 (but TCP this time rather than UDP). The data you should send is the contents of argv[0], which will be the name of your executable program.

You will receive an echo response similar to the UDP packet you got in the last assignment. However, remember that TCP can divide your data across multiple packets. Therefore, on a TCP connection, you can't simply call recv() once. Instead, you have to call it multiple times, accumulating data into a buffer. Eventually, the server will have nothing left to say, and recv() will return 0. At that point, printf the buffer and exit.

Additionally, you will need to choose a non-blocking IO technique and use that. While waiting for data to be available from the socket, print one dot to stdout every 100 milliseconds. Since this program is so simple, it won't have anything useful to do during those 100ms, so you can just sleep--on Windows use Sleep(100) (from WinBase.h); on Linux usleep(100000) (from unistd.h); or, in C++11, std::this_thread_sleep::sleep_for(std::chrono::milliseconds(100)) works on both platforms (#include <chrono> and <thread>).

Linux and Windows

You will need to create two versions of the same program for this assignment: one for Linux, and one for Windows. The Windows half of your assignment should compile in Visual Studio 2017, just as before. The Linux half should consist of a makefile that builds your program via Linux Mint's default version of gcc. Using conditional compilation (e.g., "`#ifdef WIN32`") so that the same source code files compile on both Linux and Windows.

Testing

The server you'll talk to for this assignment is currently running, so you should be able to test your code at any time. If you can't get a response from the server, first see if the server received your packet: In a web browser, go to http://echo.cs260.net/packets. That web page will report all the packet activity the server has seen from your IP address, for both UDP and TCP.

For TCP activity, the server expects to see you send a FIN bit (via shutdown(SD_SEND)) and to continue receiving until the server has sent its own FIN bit. If you fail to do either of those, the web page will tell you so.

If the packet log on the web server says "not found", then the server hasn't seen any packets from you. In that case, try hitting the server with PacketSender (http://packetsender.com). If that fails, then send me an email and I'll investigate--it's possible the error lies with the server, not with your client. Note that PacketSender will not wait for more than one packet in response, so it won't show you the full response that your program will receive from the server.

Submission

Write the Windows side of your program and make sure it compiles and runs successfully as a release build. Then use the cross-platform #ifdef techniques we discussed in class to make the same source compile on both platforms. Your Linux project must have a makefile and compile with nothing installed except build-essential.

Once finished, delete the build detritus (executable files, object files, symbol tables, etc.) from both your Windows and Linux versions, then zip up what's left and upload it to Moodle.

Grading

From a base score of 100:

- If your program can't be compiled in release mode with a default installation of Visual Studio, you lose 50 points. Don't rely on any external tools such as cmake.
- If your program can't be compiled with make on Linux Mint, with nothing but build-essential installed, you lose 50 points.
- If your program uses two complete copies of the source code rather than cross-compiling with #ifdef, you lose 10 points.
- If your program doesn't connect to the server, you lose 70 points.
- If the packet you send to the server is more or fewer bytes than the name of your executable file (the contents of argv[0], or the filename stripped out of that; either is acceptable), you'll lose 20 points. Sending the entire buffer, or sending a string with its null terminator, are common mistakes you should watch out for.
- If you send multiple packets to the server, you'll lose 20 points. This will be on top of the previous penalty if applicable.
- If your program doesn't receive and print out the server's *complete* response, you'll lose 30 points. Be warned that the server deliberately stalls its response, spreading it across multiple packets with long gaps between them. You'll have to call recv() multiple times, accumulating the bytes read in a buffer, until you get back a 0.
- While your program is waiting for the server response, it should print one period character every 100 milliseconds. Thus, the final correct output will be a string of dots followed by the server's full response. The server's timing will be consistent, and grading will be done on a machine with a fast network connection, so there is an expected number of dots that should appear. (The expected “number” is actually a small range, to account for jitter and fencepost conditions.) You will lose one point for every dot you are off from this target.
- If your program prints out the server's response incompletely, or prints more bytes beyond the end of the response, you lose 10 points. Again, it's very common to allocate a buffer to accumulate the response packets into and then fail to null-terminate it before calling printf; don't do that.

- If you include build products like object or executable files in your submission, you lose 10 points.

If you score 89 or fewer points, you may correct errors and resubmit. Each resubmission carries a 10 point penalty.