CS 2690 Programming Assignment 2
Using Microsoft Visual C++ and Windows 7, 8.1 or 10

Windows Sockets TCP Echo Server in C

Approximate time required: 3 - 4 hours

Introduction

For your second programming assignment, you will create an IPv6 server console application in C that uses Windows Sockets to echo a text message back to your echo client. This will be your version of the WSEchoServerv6 that you used to test your WSEchoClientv6. You will need a working version of your WSEchoClientv6 (Program 1) for testing. No program template/outline will be provided for this assignment. You should be able to use your WSEchoClientv6 as a starting point. All related files can be found in Canvas under `/Programming Assignments/Program 2/`.

As before, you will use the Microsoft Visual C++ compiler in one of the recent versions of Visual Studio. If you don't have access to a Windows computer, you can use one of the computers in the Network Lab (CS 516) to code, compile and test. Any UVU computer lab that has Visual Studio can be used for coding, but only CS 516 has all of the required test software. For testing, we will run both client and server on the same computer again, in loopback mode. You will need to run Wireshark v3.0.x on the same computer to analyze the packet traffic. (That should be the Wireshark version that you used for Lab 1.)

You will want to have the PowerPoint slides available that discuss TCP and server-side Sockets programming (mainly Lecture 12 and a few slides from Lecture 11). You will find the Sockets discussion on textbook pages 36 – 44 generally helpful, but bear in mind that the C code examples in the text use Berkeley Sockets, which is somewhat different than Windows Sockets. Use the PowerPoint slides for specific Windows Sockets code examples.

This assignment should not take as long as Program 1, however you may find the new functionality relating to creating and binding of the server socket, and the logic for the call to `accept()` to be challenging.

(CONTINUED ON NEXT PAGE)

Evaluation of Programming Assignments

The three programming assignments are worth a total of 10% toward your semester grade. Each programming assignment will be worth 10 points. Your submission will be evaluated as follows:

Program meets all specifications	10 points
Program works properly, but does not meet all specifications	9
Program is generally correct, has minor functional problems	7 – 8
Part of the program is correct, but has major functional problems	4 – 6
Partially correct source code, does not compile (no .exe submitted), or source code is difficult to read	1 – 3
Source code doesn't match .exe, or is unreadable	0
Program is virtually the same as another student's submission or otherwise plagiarized	0

Program Specifications

Name your source code files `WSEchoServerv6.c` and `ProcessClient.c`. Use the `DisplayFatalErr.c` module from Program 1 to handle errors. Use your own `WSEchoClientv6.exe` (Program 1) to test your server.

This program will be implemented as a command line program in C. It will obtain its input data from one command line argument, as follows:

```
WSEchoServerv6 <server port>
```

For example:

```
WSEchoServerv6 5000
```

You may design your server to use a default port number if none is provided as a command line argument. When starting your server, use a port number *above* the range of well-known port numbers (1 – 1024).

All network communication will be achieved using the Windows Sockets API. Do not use C++ socket classes.

Include the program number, and your name, section, date, Windows version and Visual Studio version in the program header comments. The comments containing the declaration on cheating must be included.

Include the header (`.h`) files required to support Windows Sockets.

Use C or C++ style comments to document each call to the WinSock API. Refer to `Programming Assignments/Program 1/style.pdf` in Canvas for style guidelines.

Feel free to insert additional `printf()` and `getchar()` statements into your server code for debugging purposes, *but you must remove them and recompile before submitting your program to Canvas,* so that the grader doesn't have to press ENTER repeatedly to get your server to accept a client connection and echo the message. Failing to do this may lower your score.

Code the following required procedural activities in the order listed.

1. The primary source code file will be named `WSEchoServerv6.c`. This will contain your `main()` function. As with your client, verify the correct number of command line arguments have been provided by the user. You don't need to validate the content of those arguments. If no port number is included on the command line, you may use a default port of your own choosing.

2. Initialize the WinSock DLL. After a successful call to WSAStartup(), handle any errors by calling `DisplayFatalErr()`.

3. Create the server socket.

4. Load the server information into the server's `sockaddr_in6` structure (see Lecture 12 slide titled "Server Socket Initialization"). Use `in6addr_any` as the server's IP address.

5. `bind()` the server socket to this `sockaddr_in6` structure.

6. Call `listen()` to tell the server the maximum simultaneous client connection requests to allow.

7. Display a message on the server console that includes your initials, similar to this:

   ```
   JD's IPv6 echo server is ready for client connection...
   ```

8. Enter a "forever" loop like this:

   ```
   for (;;) {...}
   ```

   From within this loop, call `accept()` and wait for a client connection. Calling `accept()` from within this for loop will allow the server to accept connection requests from one or more clients, but only one at a time. Echo back one complete client message per connection request. (This non-threaded program will not support multiple simultaneous client connections.)

9. Each time a client connects to the server in this loop, display the IP address and port number of the client, and the server's own port number (from the server's `sockaddr_in6`) on the server's console, as shown below:

   ```
   Processing the client at <IP addr>, client port <port>, server
   port <port>
   ```

   For example:

   ```
   Processing the client at ::1, client port 54321, server
   port 5000
   ```

   Use `inet_ntop()` to convert an IP address from network format into a text string suitable for display (as shown above). Use `ntohs()` to convert a 16-bit port number from network format back into a 16-bit Windows integer. Verify that both port numbers are displayed as numerically correct values (not reversed).

10. From within the “forever” loop, call a function named `ProcesClient()` to receive the message from your client. Implement `ProcesClient()` in a *separate source code file* named `ProcessClient.c`. Declare `ProcessClient()` in `WSEchoServerv6.c`, outside of `main()`, the same way that `DisplayFatalError()` is declared. Check for errors in the Sockets receive process, as indicated by a negative return value from `recv()`.

11. Echo the message back to the client from within `ProcessClient()`. *Be sure that all of the client’s message has been received and echoed by the server.* You can’t count bytes as you did with the client, because the server doesn’t know in advance how many bytes to expect from the client, and whether *all* of those bytes will be received in the initial call to `recv()`. So call `recv()` repeatedly until `recv()` returns message length 0, indicating that the client has closed the connection. (See the Lecture 11 slide titled “recv( )” for details.)

12. While still in `ProcessClient()`, close the *client* socket, return to the “forever” loop in `WSEchoServerv6.c`, then call `accept()` and wait for another client connection.

13. While waiting in the “forever” loop for client communication, the server can be terminated with CTRL+C. No special programming is required to enable this. Note that CTRL+C works when running the server from the command line, but not if you are running it in the Visual Studio debugger. Because we are using this crude termination mechanism for the server, no call to `closesocket()` will be needed to close the *server* socket and no call to `WSACleanup()` will be needed to release the WinSock DLL resources. Those functions will be handled by the operating system. (In a real sockets program, omitting `closesocket()` and `WSACleanup()` would be considered poor design.)

Testing Your Echo Server

Design and code your echo server, build the solution and debug any compiler errors. After your program compiles successfully, test it with your client in loopback mode, running both programs from the command line. Start the server first, like this:

```
WSEchoServerv6 <server port>
```

In a second command line window, run the client like this:

```
WSEchoClientv6 ::1 <server port> <"Message to echo">
```

Verify that your server echoes the message back to your client, and that the client can reconnect and echo another message while the server continues running. If not, debug the server.

Capturing IPv6 Loopback Traffic with the Wireshark Npcap Driver

Next, we are going to use Wireshark (v3.0 or newer) to capture the TCP/IPv6 loopback packets transmitted between your client and server. This capability, provided by the Npcap driver, is new to Wireshark and possibly still a bit buggy. Familiarize yourself with steps 1 – 5 below before you try them, so that you can execute them fairly quickly.

1. Restart your `WSEchoServerv6` in a command line window, this time using port 1114.

2. Start Wireshark v3.0.x. Select **View | Name Resolution** and check **Resolve Transport Addresses**. This will cause Wireshark to display names of application programs associated well known or registered port numbers.

   Click the **Capture** tab **| Options**, and locate the *Npcap Loopback Adapter*. Don't start the packet capture yet. (If you don't see a *Npcap Loopback Adapter*, read the **Debugging Npcap Loopback Capture Failure** section below.)

3. In a second command line window, enter something like the following to run your client (but don't execute it yet):

   ```
   WSEchoClientv6 ::1 1114 "NOT Mini-SQL!"
   ```

4. In Wireshark, select the *Npcap Loopback Adapter* and start the Wireshark capture. Then execute the client, send and receive your message, and let the client disconnect. Stop the Wireshark capture.

5. Enter and enable the following Wireshark display filter:

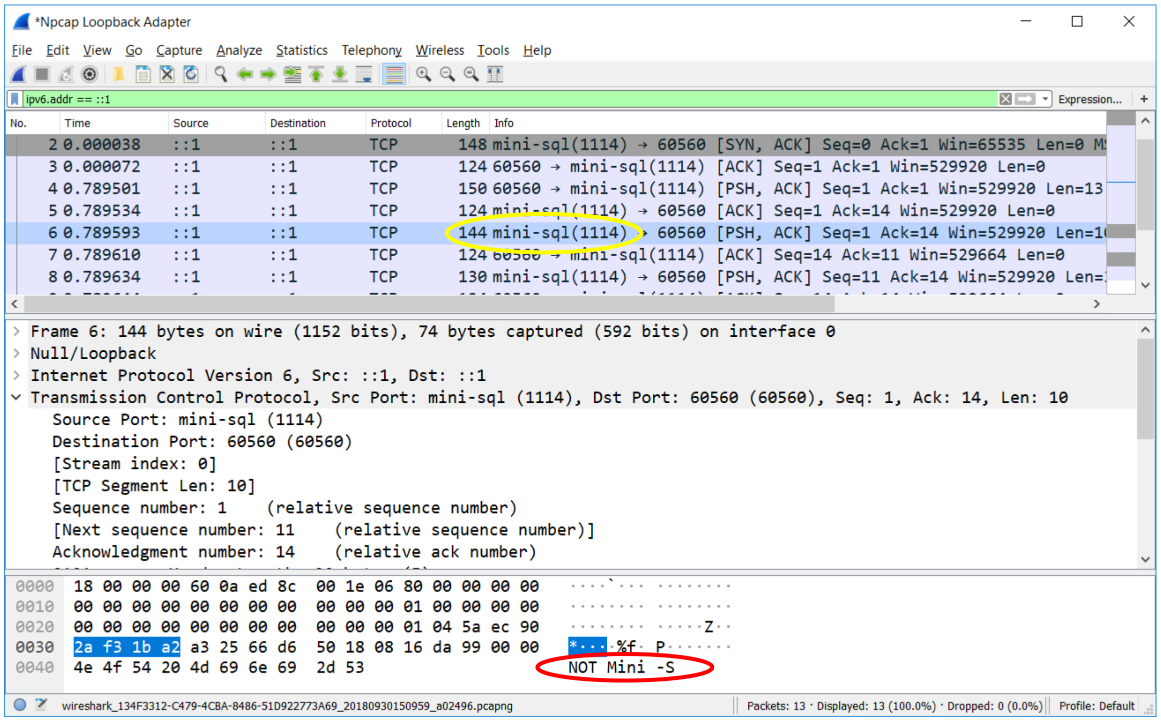
   ```
   ipv6.addr == ::1
   ```

   If you don't see any IPv6 loopback packets in the *Packet Details* pane, see the **Debugging Npcap Loopback Capture Failure** section below.

In Wireshark, scroll down so that the first (or only) packet containing your message being echoed back *from your server* is displayed in the middle *Packet Details* pane. You should see your message (or the beginning of it) displayed in ASCII text in the lower *Packet Bytes pane (see Figure 1 below).* Don't be surprised if this echo response contains an incomplete message.

Make a Wireshark screen capture of this packet with your message visible in the Packet Bytes pane, and paste it into a `.pdf`, `.doc`, `.docx`, `.odt`

(CONTINUED ON NEXT PAGE)


*Npcap Loopback Adapter

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ipv6.addr == ::1

```
No.  Time      Source  Destination  Protocol  Length  Info
   2 0.000038  ::1     ::1          TCP       148 mini-sql(1114) → 60560 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 M
   3 0.000072  ::1     ::1          TCP       124 60560 → mini-sql(1114) [ACK] Seq=1 Ack=1 Win=529920 Len=0
   4 0.789501  ::1     ::1          TCP       150 60560 → mini-sql(1114) [PSH, ACK] Seq=1 Ack=1 Win=529920 Len=13
   5 0.789534  ::1     ::1          TCP       124 mini-sql(1114) → 60560 [ACK] Seq=1 Ack=14 Win=529920 Len=0
   6 0.789593  ::1     ::1          TCP       144 mini-sql(1114) → 60560 [PSH, ACK] Seq=1 Ack=14 Win=529920 Len=1
   7 0.789610  ::1     ::1          TCP       124 60560 → mini-sql(1114) [ACK] Seq=14 Ack=11 Win=529664 Len=0
   8 0.789634  ::1     ::1          TCP       130 mini-sql(1114) → 60560 [PSH, ACK] Seq=11 Ack=14 Win=529920 Len=
```

```
> Frame 6: 144 bytes on wire (1152 bits), 74 bytes captured (592 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
v Transmission Control Protocol, Src Port: mini-sql (1114), Dst Port: 60560 (60560), Seq: 1, Ack: 14, Len: 10
    Source Port: mini-sql (1114)
    Destination Port: 60560 (60560)
    [Stream index: 0]
    [TCP Segment Len: 10]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 11    (relative sequence number)]
    Acknowledgment number: 14    (relative ack number)
```

```
0000  18 00 00 00 60 0a ed 8c  00 1e 06 80 00 00 00 00   ....`... ........
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 00  00 00 00 01 04 5a ec 90   ........ .....Z..
0030  2a f3 1b a2 a3 25 66 d6  50 18 08 16 da 99 00 00   *....%f. P.......
0040  4e 4f 54 20 4d 69 6e 69  2d 53                    NOT Mini -S
```

wireshark_134F3312-C479-4CBA-8486-51D922773A69_20180930150959_a02496.pcapng | Packets: 13 · Displayed: 13 (100.0%) · Dropped: 0 (0.0%) | Profile: Default

Figure 1. Sample Wireshark Screen Capture for Question 1.

Debugging Npcap Loopback Capture Failure

If you don't see any IPv6 loopback traffic in Wireshark, go to **Control Panel | Network and Internet | Change adapter settings**, and verify that the Npcap Loopback Adapter is enabled. (Read carefully, there may be several of them. If so, enable all of them.)

Then, go back to the Wireshark the **Capture** tab **| Options**, watch the Traffic graphs next to the various interfaces and run your client again. The Npcap loopback adapter may have a different name, for example *Local Area Connection 2*, *Ethernet 3* or *Adapter for loopback traffic capture*. The corresponding Link-layer Header column will usually say "BSD loopback". Try any of the interfaces that seem to be capturing traffic (indicated by a jagged Traffic graph), and see if you can locate the IPv6 loopback traffic.

If you have tried all of the possible loopback interfaces and are unable to capture your IPv6 loopback traffic, copy your client and server executables onto a flash drive. Visit the Network Lab (CS 516) and try this on one of the lab computers. (Use your regular UVU login and password.)

Making Sense Out of the Wireshark Capture

In your Wireshark capture, you will see that the TCP connection is established as usual (**SYN**, **SYN+ACK, ACK**) and terminated as usual (**FIN, ACK, FIN+ACK, ACK**). However, Wireshark seems to think that your echo client and server messages are *Mini SQL* traffic – not what we were expecting.

Question

1. Explain why Wireshark thinks that you client and server are exchanging Mini SQL messages.

Submitting Your Program to Canvas

Before submitting your program, review the Program Specifications section to confirm that your program meets the requirements. The grader will not compile your code. If your program does not execute from the Windows command line, or if an executable is not provided, you can earn no more than 30% of the possible points. If you have compiled your program in Debug mode, recompile it in Release mode before you submit it. See `Compiling a C Winsock Program in Visual Studio Community 2019.docx` page 6 for instructions on how to compile in Release mode.

Combine your source code (`*.c`), executable (`*.exe`), and any other required files into a single zip file. *Do not submit any other files and do not submit the entire Visual Studio project.* Name the zip file with the program number and your initials, for example `prog2dbh.zip`. Once you have zipped together your files, upload the zip file to Canvas. Be sure to include all the files listed in the Summary of Deliverables below.

If you have disabled your firewall, be sure to enable it after testing is complete.

Summary of Deliverables

- `WSEchoServerv6.c`
- `ProcessClient.c`
- (Your version of) `WSEchoServerv6.exe` (compiled in Release mode)
- Your Wireshark screen capture in `.pdf`, `.doc`, `.docx`, `.odt` or `.rtf` format
- Your answer to Question 1
- All of the above contained in a single zip file named `prog2***.zip`
- Do not submit `DisplayFatalErr.c`, `WSEchoClientv6.c`, `WSEchoClientv6.exe` or any other files
- Do not submit the entire Visual Studio project