Homework #4 - Socket Communication

Goals of this assignment

1. Learn, through the use of sockets, a practical application for Inter-Process Communication
2. Go outside of your coding comfort zone!
3. Maybe learn GUI development (extra credit)

Description

For this assignment you will be asked to create a chat application. For full credit you may implement it as a console application. However, I would like to challenge you to make it a Graphical User Interface (GUI). If you successfully implement the GUI you stand to get 10% points extra credit. The application will consist of the following elements:

1) A transmit socket for sending chat messages to a specified endpoint,
2) A receive socket for receiving chat messages, and
3) Optionally, a Graphical User Interface (GUI) using Microsoft Foundation Classes (MFC). (Other GUI development environments are also acceptable)

At the most basic level, the application will allow a user to enter chat messages (either on the command line or into an edit box) and send them to another process on the same computer, simulating a chat application over the Internet. For the GUI version, transmitted (and received) chat messages will appear in a larger, read-only, edit box.

Requirements and Specification

1. Messages shall be transmitted using a User Datagram Protocol/Internet Protocol (UDP/IP) Socket. Use the following link for explanation and examples of how to use socket functions in a Windows environment:

https://stackoverflow.com/questions/679145/how-to-set-up-a-winsock-udp-socket
https://msdn.microsoft.com/en-us/library/windows/desktop/ms741394(v=vs.85).aspx

(Note: in the Stack Overflow link, the selected answer uses a deprecated structure, so look at the second suggested solution from user, Chuque)

The socket functions you will need to use are socket(), sendto(), bind() and closesocket(). However, as you now know, there are multiple libraries that abstract these calls to make it “easier” to implement. You will need to use a single socket for sending, and a second socket for receiving. Use IP address 127.0.0.1 and port 3514

(CS351-Section 4) to address the other process. For the receive socket, you must receive on, or bind() to a different port - 3515. You will use the function recvfrom(), which is a blocking call (i.e. it waits for I/O). Since you will likely want to send one or more messages while waiting for a message to be received, this function must be performed in a separate Thread.

2. When the user enters a chat message and presses return (or a Send button in a GUI), the message shall be sent to the addressee. If you are using a GUI, the message shall appear in the "Received Messages" window. The format of a Chat Message shall be as follows:

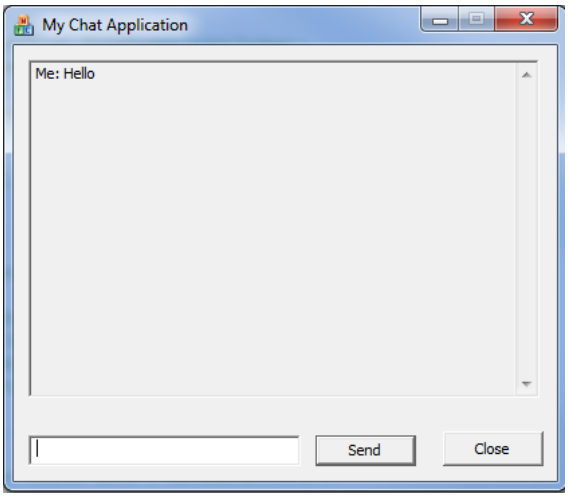
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	…..	Byte n
A_1	A_2	A_3	A_4	A_5	A_6	…..	A_n

A_1 - A(n-1): Alphanumeric characters representing the chat message entered by the user.

A_n: NULL character - '\0'

With the receive socket, upon receiving a Chat Message, your application shall write the received message to the console or add the message to the "Received Messages" window if a GUI is used.

3. I will provide a companion application (Chat App (Oates).exe) that will allow you to test yours. It will show Chat Messages in the Receive Messages window. Also, it will allow you to send Chat Messages to your application (addressed to port 3515) to test your receive socket. This is what my application window looks like. Feel free to model yours after mine or come up with your own design.



Grading

If you complete the console version of the assignment (a console application that allows a user to send chat messages to 127.0.0.1:3514) and receive chat messages on port 3515, you will get full credit.

If you make your application a GUI, you will earn an additional 10% points to be added to the homework grade.