CSC 3350 Operating Systems
HW 6: IPC Socket Talk (Final Examination/Lab)
Due: Tuesday, June 7, 2016 10:00am (or earlier). Instructor's Office

Design and implement a TALK application, as described in this Problem Statement.

- Overview
  Your application will consist of two separate programs, **SERVER.exe** and **CLIENT.exe**. Both programs will use sockets configured to use the TCP/IP protocols. The user will manually start the server and then start the client to establish a socket communication linkage.

  When the SERVER starts, it will bind itself to listen for connection requests on a system selected server-port-number. It will print out its server HostName and server-port-number to the user, and then wait for an incoming client connection request. When a client initiates a connection request, the SERVER will create and connect a server socket to communicate with the client.

  Once the SERVER is running, the CLIENT program can be started using command-line parameters to specify the server-port-number and (optional) HostName. The CLIENT must create a client socket and initiate a connection request to the SERVER, thereby establishing a socket-based VPN (virtual private network) communication path.

  Once the connection is established, both the SERVER and the CLIENT applications must engage in a "talk" session. Any message buffers received through the socket must be displayed to the user, prefixed on the screen with "<". Any lines of text typed in by the user must be displayed to the user locally, prefixed on the screen with ">", and then sent to the other process as a message-buffer using the socket.

  If the user of either the SERVER or the CLIENT types "exit", then both applications should quit talking and terminate. If either the user types "exit", or if a "exit" message is received via the socket, then terminate the talk.

  Although not required, it is possible in your design to implement one function (e.g., talk(skt)) that can be compiled and linked into both the server and the client application.

- Socket and HostName Setup and Cleanup
  Both the SERVER and the CLIENT create and initialize a SOCKET object and related SOCKADDR_IN object in similar fashion.
  - WSAStartup(…) initializes the socket package. Use version (2, 2).
  - Create a stream socket with Internet Affinity (not UNIX Affinity).
  - Create and initialize a HostName SOCKADDR_IN object, used to reference the SERVER Host listener.
    - The HostName.sin_family field must specify Internet Affinity.
    - The HostName.sin_addr field must specify the SERVER address information. Use
      LPHOSTENT host = gethostbyname(serverHostName);
      to obtain a pointer to a host information object about the server-host. Then you can copy the host h_addr_list table to the name.sin_addr field
      CopyMemory(&HostName.sin_addr,
      host->h_addr_list[0], host->h_length);

    - The HostName.sin_port field must specify the integer SERVER portNumber. For the SERVER application, initially set portNumber to zero so that the system will select an unused port number. For the CLIENT application, portNumber is specified on the command-line.

        - Since the SOCKET package is implemented as transportable package, byte-ordering for internal multi-byte values (like portNumber) becomes an issue.
            - Anytime you specify a host port-number to the Socket package, you must use the host-to-namespace function to perform any necessary conversions:
                ns_value = htons((u_short) HostspacePortNumber)

            - Similarly, anytime you obtain a port-number value back from the Socket package, you must use the name-to-hostspace function to perform any necessary conversions:
                hs_value = ntohs((u_short) NamespacePortNumber)
    - The SERVER uses the bind(…) call to link a socket to the HostName object, and then listen(…) and accept(…) to create a new socket that connects to the client.
    - The CLIENT uses the connect(…) call to link a socket to the HostName object.
    - Be sure to close sockets when you are finished with them.
    - Call WSACleanup(); at the end.

- **SERVER.exe**
  Once a socket and a HostName object are setup, the SERVER uses the bind(…) call to link the socket to the HostName object. Then the SERVER must print out on the screen a short message with it's serverHostName and the port-number that it is listening to. Use getsockname(…) with the socket and the HostName parameters to update the HostName fields with the specific information from the binding operation. The sin_port field contains the Namespace_portnumber that is in use.

  Then the SERVER registers itself as a listener, and uses the accept (…) function to wait for a connection request from a client. Note that accept(…) returns a new socket handle that is to be used in communicating with the client application. For this lab, the original socket (that was used for the listening) is not needed after connecting with a client, and should be closed.

- **CLIENT.exe <portNumber> [<serverHostName>]**
  <portNumber> is a (u_short) integer value representing the Hostspace_portnumber.
  If <serverHostName> is not specified, then CLIENT assumes that the local machine is running the SERVER application, and uses 'gethostname(…)' to determine the serverHostName string value.

  Once a socket and a HostName object are setup, use the connect(…) function to establish the VPN connection between the CLIENT and the SERVER sockets.

- Be sure to **"#include <winsock.h>"** in your code, and to modify the Visual Studio project configuration property settings to add the **"wsock32.lib"** library to the list of Linker Inputs.

- To assist you in your testing, sample executable versions of the two applications are available with the lab assignment on Blackboard: **TalkServer.exe** and **TalkClient.exe**. Note that this lab requires you to implement your versions of both of these applications.

Turn in:

- ZIP file containing the **server.exe**, **client.exe**, and the entire Visual Studio project(s) for the lab. Be sure to Build/Clean and delete the large .sdf project file before ZIPPing.
  Submit the ZIP using Blackboard file attachment for this lab.
- Printed source code listings.
- Printed Lab summary report:
    - Include instructions for running your lab.
    - Include a brief 1-2 paragraph summary statement that describes how well the lab works, and what you learned in doing the lab.