THE PRIMARY COMMUNICATION MECHANISM FOR INTERNET APPLICATIONS, A SOCKET IS A VIRTUAL CONNECTION BETWEEN TWO APPLICATIONS, PERMITTING COMMUNICATION BETWEEN TWO PROCESSES. A SOCKET IS CREATED DYNAMICALLY AND PROVIDES FOR BIDIRECTIONAL TRANSMISSION.


SERVER

SERVER COMPLETES SOCKET, BIND, AND LISTEN OPERATIONS.

CLIENT

CLIENT NOT YET ACTIVATED.

SERVER

SERVER AWAITS CLIENT CONNECT REQUEST.

CLIENT

CLIENT USER SPECIFIES SERVER’S IP ADDRESS AND APPROPRIATE PORT NUMBER AND ESTABLISHES CLIENT SOCKET.

SERVER

SERVER ACCEPTS CONNECTION, SETS UP TEMPORARY SOCKET TO RECEIVE WHILE IT CONTINUES TO LISTEN.

CLIENT

CLIENT SENDS ITS DATA.

SERVER

SERVER RECEIVES CLIENT’S DATA.

CLIENT

CLIENT AWAITS SERVER’S DATA.

SERVER

SERVER SENDS ITS DATA.

CLIENT

CLIENT RECEIVES SERVER’S DATA.

SERVER

SERVER CONTINUES TO LISTEN.

CLIENT

CLIENT CLOSES ITS SOCKET.
To better illustrate the socket communication process, you will enhance these two programs to include features that are missing from the provided code:

1. While the server is set up to establish a connection with anyone running the client program (as long as they correctly identify the server’s IP address and the selected port number: #10500), the server’s output gives no indication regarding the client with which it is currently communicating. As part of your assignment, change the client code so that it begins by querying the user for an ID number, and instead of sending a user-specified message to the server, the client will transmit its ID number, which will, in turn, become part of the server’s output whenever it receives a message from the client.

2. The original program does not illustrate how the client handles a sequence of connections to a client. As part of your assignment, you will alter the client code to loop through a sequence of connections and disconnections, separated by one-second intervals. Each time the client reconnects, it will attach an iteration value to its ID number, which will then be sent to the server. For example, if the client ID number is specified as 637, then client 637 will begin by connecting with the server, sending its “637.00” message, and then disconnecting. Client 637 will then delay one second, reconnect with the server, send its “637.01” messages, and then disconnect. This will continue for 30 seconds, after which client 637 will permanently disconnect.

3. Notice that the server’s read/write loop and the client’s write/read loop are not implemented in the original program. Instead, a single handshaking transaction takes place. In your enhanced version of this program, you will implement both of these loops. The client will loop a random number of times (between 1 and 10), and the message will include a Greek alphabet suffix that indicates the index of the transaction being sent to the server. Thus, when client 637 is working on its first connection to the server (say, with 5 transactions), it will send “637.00.alpha”, “637.00.beta”, “637.00.gamma”, “637.00.delta”, and “637.00.epsilon” messages. When client 637 is working on its second connection to the server (say, with 2 transactions), it sends a “637.01.alpha” message, followed by a “637.01.beta” message. With its third connection (say, with 7 transactions), the messages will be “637.02.alpha”, “637.02.beta”, “637.02.gamma”, “637.02.delta”, “637.02.epsilon”, “637.02.zeta”, and “637.02.eta”. (Use the sequence “alpha”, “beta”, “gamma”, “delta”, “epsilon”, “zeta”, “eta”, “theta”, “iota”, “kappa” for this purpose.)

4. To give the server some control over terminating the connection with the client, the server’s read/write loop will put a limit on how many transmissions it allows from a particular client (another randomly generated number between 1 and 10), closing the connection when the client exceeds this number of transaction attempts. Thus, when client 637 is working on its 29th connection to the server, the server may limit the transactions to eight while the client is limiting the transactions to six, in which case the client successfully sends all six of its messages (“637.29.alpha”, “637.29.beta”, “637.29.gamma”, “637.29.delta”, “637.29.epsilon”, “637.29.zeta”), with the server returning a timestamp for each message. However, when client 637 is working on its 30th connection to the server, the server may limit the transactions to two while the client is limiting its transactions to five, so the client successfully sends its first two messages (“637.30.alpha” and “637.30.beta”), but the server closes the connection when the third message (“637.30.gamma”) is attempted.

The output for the client and server programs is displayed on the next page (with the time limit reduced to four seconds just for illustration purposes), and executable versions for both programs are part of the zipped package that is available on the course Web site.

Keep your code modular and readable, with explanatory comments inserted as needed (i.e., explanatory paragraphs at the top of each program file, explanatory sentences prior to each function implementation, and in-line comments only where essential within complicated code). Clearly mark all code that you add or alter in the original programs, and comment out all original program code that you determine needs to be deleted or replaced.
MAKE SURE THAT YOUR CLIENT AND SERVER PROGRAMS HAVE BEEN SET UP TO LINK TO THE WS2_32.LIB LIBRARY, AS ILLUSTRATED BELOW.

This assignment is due on your drop-box by 3:00 PM on Tuesday, February 7, 2012.