**Winsock Exercises 2**

**Exercise 2**. (Echo)

Write a stream based echo server printing out the message received from the client, echoing it back, until the client closes the connection.

Write a stream based echo client sending messages to the echo server, receiving each message returned by the server. Terminate the connection when “quit” is entered.

Hint: modify the presented stream based TCP client (TCPClient.cpp) and server programs (TCPServer.cpp) to transfer multiple messages back and forth (until the client terminates the connection).

Client side (project name:EchoClient)

char sendbuf[DEFAULT\_BUFLEN];

// Loop until “quit” is entered

while(1)

{

// Type the message

gets\_s(sendbuf);

// Bail out if "quit" is entered

if (strcmp(sendbuf, "quit") == 0)

break;

//send the message to the echo server

send(ConnectSocket, sendbuf, int(strlen(sendbuf)+1), 0);

// Receive from the server and print the message on the screen

recv(ConnectSocket, recvbuf, recvbuflen, 0);

printf("Received: %s \n", recvbuf);

}

Server side (project name:EchoServer)

// Loop until client terminates connection

do

{

// Receive from the client, and bail out if client shut down

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);

if (iResult > 0)

{

printf("Received: %s \n", recvbuf);

// Echo the buffer back to the sender

send( ClientSocket, recvbuf, iResult, 0 );

}

else if (iResult == 0)

printf("Connection closing...\n");

else {

printf("recv failed with error: %d \n", WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

return 1;

}

} while (iResult > 0);

**Exercise 2a**. (EchoServer2)

Modify your solution to Exercise 2 to write a stream based echo server, which can simultaneously handle multiple clients connecting to it. No modification of the client code is necessary, but multiple instances of the client should be started. Hint: use Windows threads functions.

#include <string.h> // Needed for memcpy() and strcpy()

#include <process.h> // Needed for \_beginthread() and \_endthread()

Add these lines before main()

//----- Globals ---------------------------------------------------------------

int Count; // Thread counter

//----- Function prototypes -------------------------------------------------

void do\_service(void \*client\_s); // Thread function

In the main () function add the following variables.

unsigned int client\_s; // Client socket descriptor

struct sockaddr\_in client\_addr; // Client Internet address

struct in\_addr client\_ip\_addr; // Client IP address

int addr\_len; // Internet address length

char ipstringbuffer[46];

Add the following lines after a socket is created and is put to listening state.

// Main loop (Loop forever)

Count = 0;

while (1)

{

Count++;

printf("Count=%d \n",Count);

// Accept a connection. The accept() will block and then return with

// client\_addr filled-in.

addr\_len = sizeof(client\_addr);

client\_s = accept(ListenSocket, (struct sockaddr \*)&client\_addr, &addr\_len);

// Copy the four-byte client IP address into an IP address structure

// - See winsock.h for a description of struct in\_addr

memcpy(&client\_ip\_addr, &client\_addr.sin\_addr.s\_addr, 4);

// Print an informational message that accept completed

printf("Connection %d accepted!!! \n", Count);

inet\_ntop(AF\_INET, &client\_ip\_addr, ipstringbuffer, sizeof(ipstringbuffer));

printf("\tClient socket number: %d\n", client\_s);

printf("\tIPv4 address: %s\n", ipstringbuffer);

printf("\tPort nuber: %d\n", ntohs(client\_addr.sin\_port));

if (\_beginthread(do\_service, 4096, (void \*)client\_s) < 0)

{

printf("ERROR - Unable to create thread \n");

exit(1);

}

}

// Never reached!!!

// Wait for all threads to finish

while(Count);

// Close open sockets

closesocket(ListenSocket);

// This stuff cleans-up winsock

WSACleanup();

}

//===========================================================================

//= Thread function to service a single client =

//===========================================================================

void do\_service(void \*client\_s)

{

char in\_buf[1024]; // Input buffer for response

printf("thread beninging... \n");

// Loop until client shut down

while(1)

{

// Receive from the client

if (recv((unsigned int)client\_s, in\_buf, sizeof(in\_buf), 0) == 0)

break; // when client shut down

printf("Received from client... data = '%s' \n", in\_buf);

// Echo the received message to the client

send((unsigned int)client\_s, in\_buf, (strlen(in\_buf) + 1), 0);

}

printf("thread completed... \n");

// Decrement for a completed thread

Count--;

// Close all open sockets and end the thread

closesocket((unsigned int)client\_s);

\_endthread();

}

Hand in

1. EchoClient.cpp, EchoServer.cpp, EchoServer2.cpp
2. Echo.pdf showing (with screen shots)
   1. The execution of EchoClient and EchoServer
   2. The execution of EchoServer2 with multiple echo clients
   3. Other interesting results you wish to show