Winsock Exercises 2

Echo-Client and Echo-Server

Contents

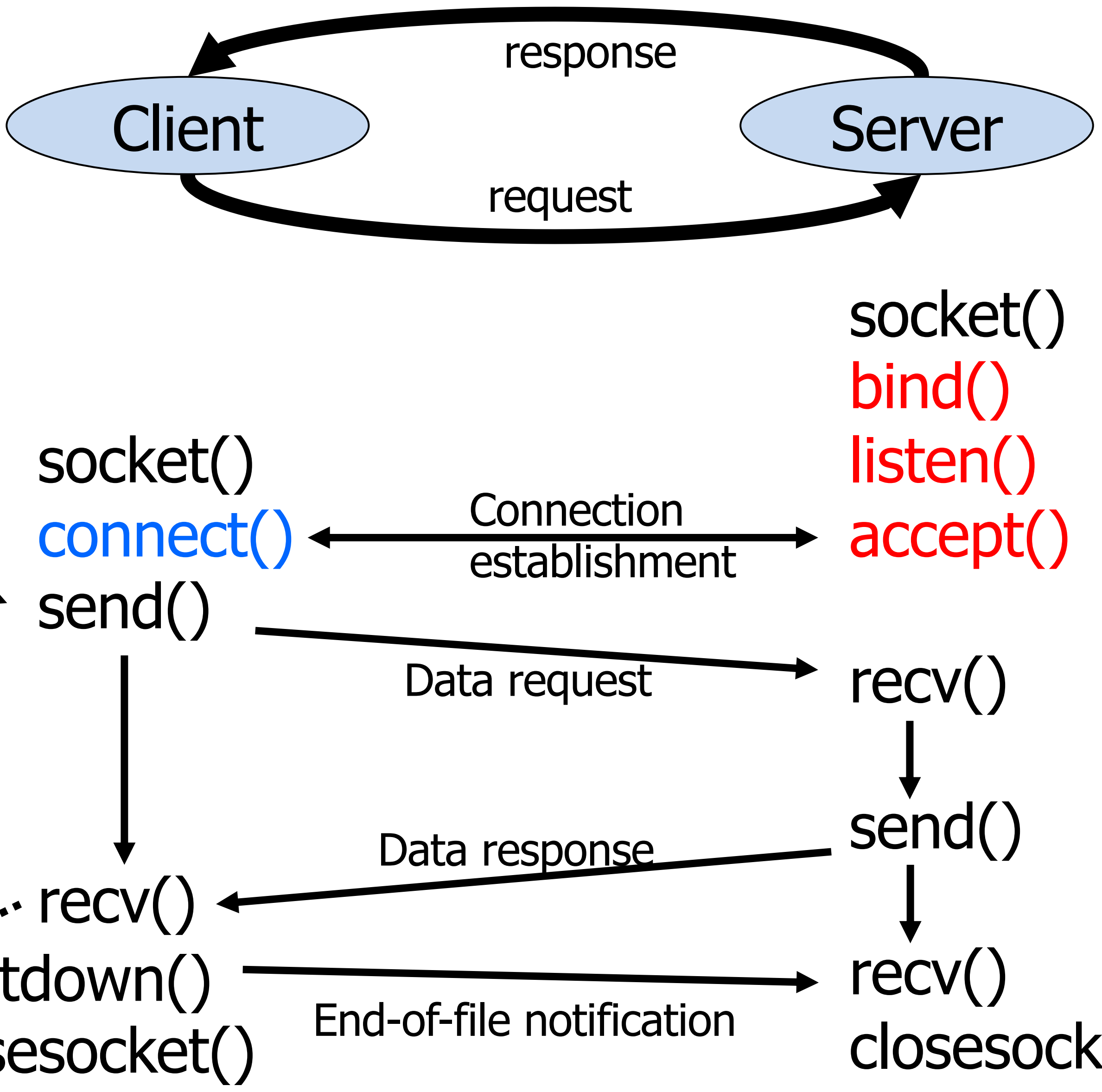
- E2: Simple Echo-client and Echo-server
- E2a: Echo-server handling multiple clients

A simple echo-client and echo-server

- Write a stream based echo server printing out the message received from the client, then 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.

Simple Client-Server Example



TCP Client Review

1. Initialize Winsock: **WSAStartup( )**
2. Create a socket: **socket( )**
3. Connect to the server: **connect ( )**
4. Send and receive data: **send( ), recv ( )**
5. Disconnect. **shutdown( )**, **closesocket( ), WSACleanup ( )**

The complete client file: TCPClient.cpp

(note: for a successful connection from a client, a TCPserver application must be running)

Echo-client

Modify TCPClient.cpp

1. Initialize Winsock. (no change)
2. Create a socket. (no change)
3. Connect to the server. (no change)
4. Send and receive data. (minor change)
5. Disconnect. (no change)

Echo-client: send and receive data

```
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);
  }
```

TCP Server Review

1. Initialize Winsock: WSAStartup( )
2. Create a socket: socket ( )
3. Bind the socket: bind ( )
4. Listen on the socket for a client: listen( )
5. Accept a connection from a client: accept( )
6. Receive and send data: recv( ), send( )
7. Disconnect: shutdown( ), closesocket( )

The complete server file: TCPServer.cpp

Echo-server

Modify TCPServer.cpp

1. Initialize Winsock (no change)
2. Create a socket (no change)
3. Bind the socket (no change)
4. Listen on the socket for a client (no change)
5. Accept a connection from a client (no change)
6. Receive and send data (minor change)
7. Disconnect (no change)

Echo-server: minor change to TCPServer.cpp

```
// 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);
```

Contents

- E2: Simple Echo-client and Echo-server
- E2a: EchoServer handling multiple clients

EchoServer2: handling multiple clients

- Modify your solution to Exercise 2 to write a stream based echo server, which can simultaneously handle multiple clients connecting to it.
  - Use project name EchoServer2, source file EchoServer2.cpp
  - Modify EchoServer.cpp or TCPServer.cpp
- Hint: use Windows threads functions.

- No modification of the client code is necessary, but multiple instances of the client should be started.

EchoServer2: handling multiple clients

Modify EchoServer.cpp as follows:

```
#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];
```

EchoServer2: handling multiple clients

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

```
 Count = 0; //number of thread
 while (1) // Main loop (Loop forever)
 {
   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);
   }
 }
 while(Count); // Never reached!!! // Wait for all threads to finish
 closesocket(ListenSocket); // Close open sockets
 WSACleanup(); // This stuff cleans-up winsock
}
```

Thread function to serve a single client

```
void do_service(void *client_s)
{
  char                  out_buf[1024];    // Output buffer for GET request
  char                  in_buf[1024];     // Input buffer for response
  unsigned int          retcode;              // Return code
  unsigned int          i;                    // Loop counter

  printf("thread beninning... \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();
}
```

More about Thread

MSDN Library https://msdn.microsoft.com/en-us/library/ms123401.aspx

```
uintptr_t _beginthread( // NATIVE CODE
   void( __cdecl *start_address )( void * ),
   unsigned stack_size,
   void *arglist
);

uintptr_t _beginthread( // MANAGED CODE
   void( __clrcall *start_address )( void * ),
   unsigned stack_size,
   void *arglist
);
```

Reference

- Install Microsoft Visual Studio Community 2017
  - https://www.visualstudio.com/zh-hans/downloads/

- Getting started with Winsock
  - https://msdn.microsoft.com/en-us/library/ms738545(v=vs.85).aspx

- Winsock reference
  - https://msdn.microsoft.com/en-us/library/ms741416(v=vs.85).aspx