Winsock Exercises 3

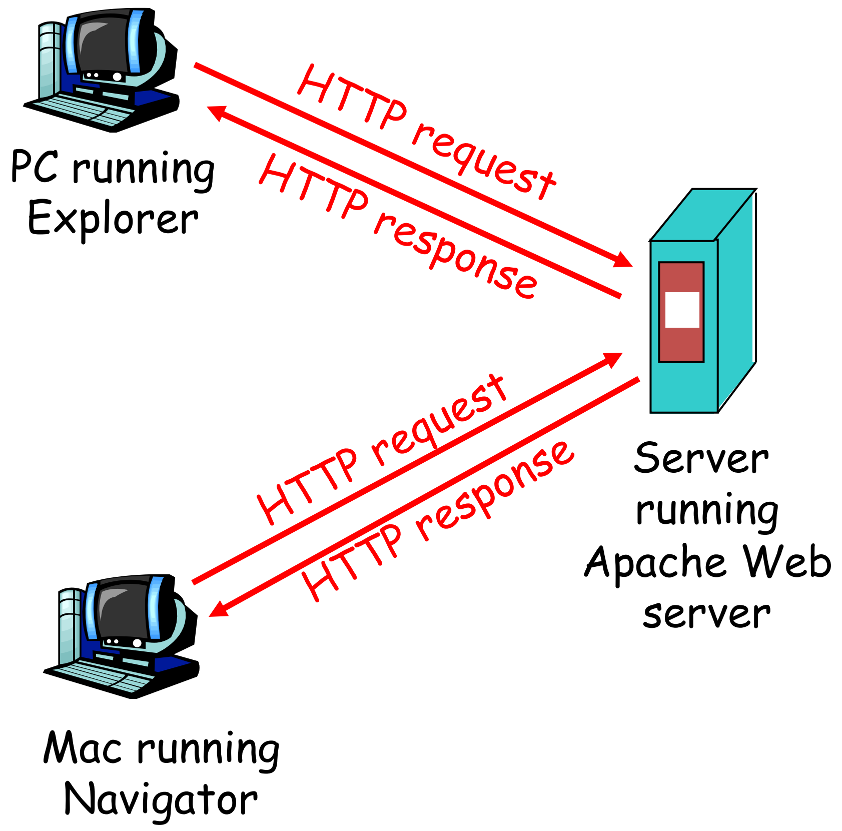
More details in WinsockExercises.docx

Contents

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser
  - *server:* Web server



HTTP overview (continued)

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages exchanged
- TCP connection closed

History

- World-Wide Web since 1990
- HTML2.0: RFC 1866 (1995)
- HTTP 1.0: RFC 1945 (1996)
- HTTP 1.1: RFC 2068 (1997)
- HTTP 2: RFC 7540 (2015)

HTTP connections

Nonpersistent HTTP

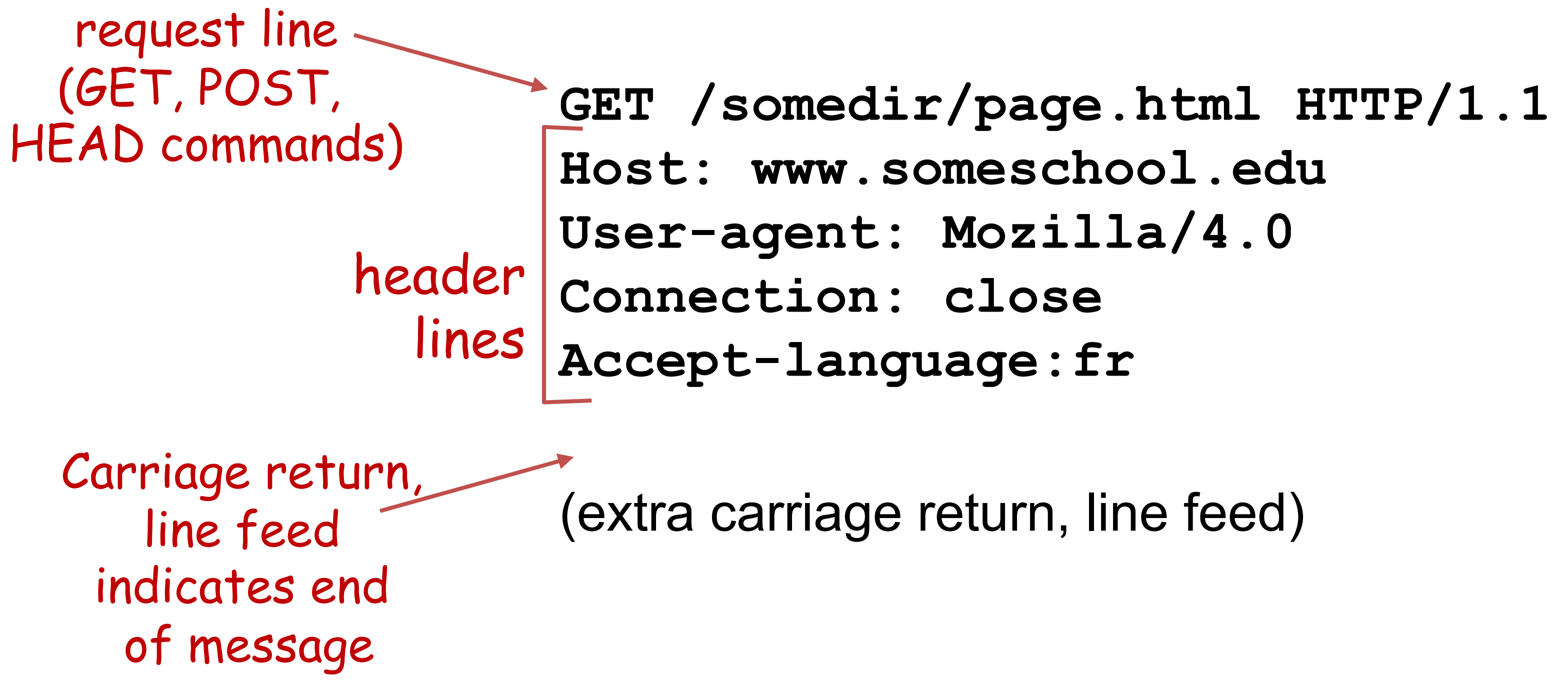
- At most one object is sent over a TCP connection.
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)


request line (GET, POST, HEAD commands)
header lines
Carriage return, line feed indicates end of message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr

(extra carriage return, line feed)
```

Method types

HTTP/1.0

- GET
- POST
- HEAD
  - Only message header fields are returned

HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

HTTP response status codes

In first line in server->client response message.
A few sample codes:

`200 OK`

- request succeeded, requested object later in this message

`301 Moved Permanently`

- requested object moved, new location specified later in this message (Location:)

`400 Bad Request`

- request message not understood by server

`404 Not Found`

- requested document not found on this server

`505 HTTP Version Not Supported`

A simple WebClient

Write a client program to execute a single HTTP GET to a Web server.

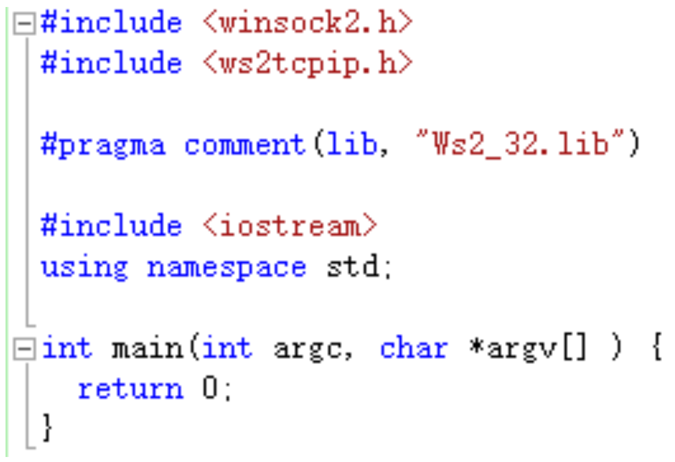
A web client performs the following steps:

- Initializes Winsock.
- Creates a socket.
- Connects to the server.
- Sends and receives data.
- Disconnects.

A simple WebClient

- Create a new project: WebClient
- Source Files →Add New Item → C++ File → WebClient.cpp

1. create a basic Winsock application



```
#include <winsock2.h>
#include <ws2tcpip.h>

#pragma comment(lib, "Ws2_32.lib")

#include <iostream>
using namespace std;

int main(int argc, char *argv[] ) {
    return 0;
}
```

A simple WebClient

2. initialize Winsock and initiate use of WS2_32.dll

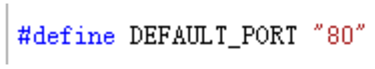
```
WORD wVersion = MAKEWORD(2, 2); // Used to request version 2.2 of Windows sockets
WSADATA wsaData;                // Data loaded by WSAStartup
int iResult;                    // Error check if WSAStartup successful

// Initialize Winsock
iResult = WSAStartup(wVersion, &wsaData);
if (iResult != 0) {
   cout << "WSAStartup failed: " << iResult << endl;
   return 1;
}
```

3. Declare an addrinfo object that contains a sockaddr structure

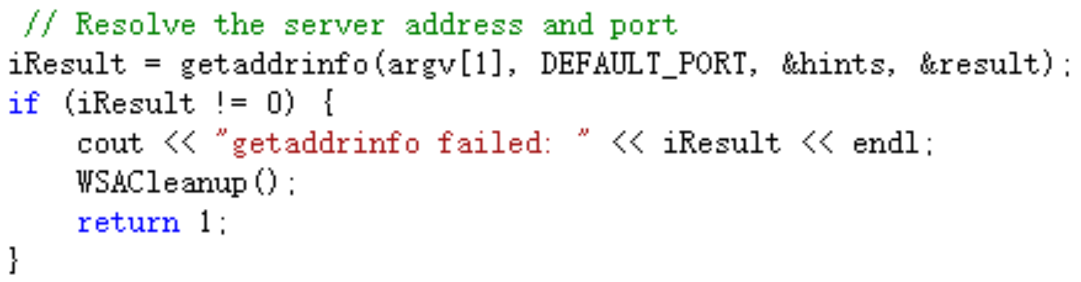
```
struct addrinfo *result = NULL,
                *ptr = NULL,
                hints;

ZeroMemory( &hints, sizeof(hints) );
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
```

A simple WebClient

4. Call the getaddrinfo function requesting the IP address for the server name passed on the command line.

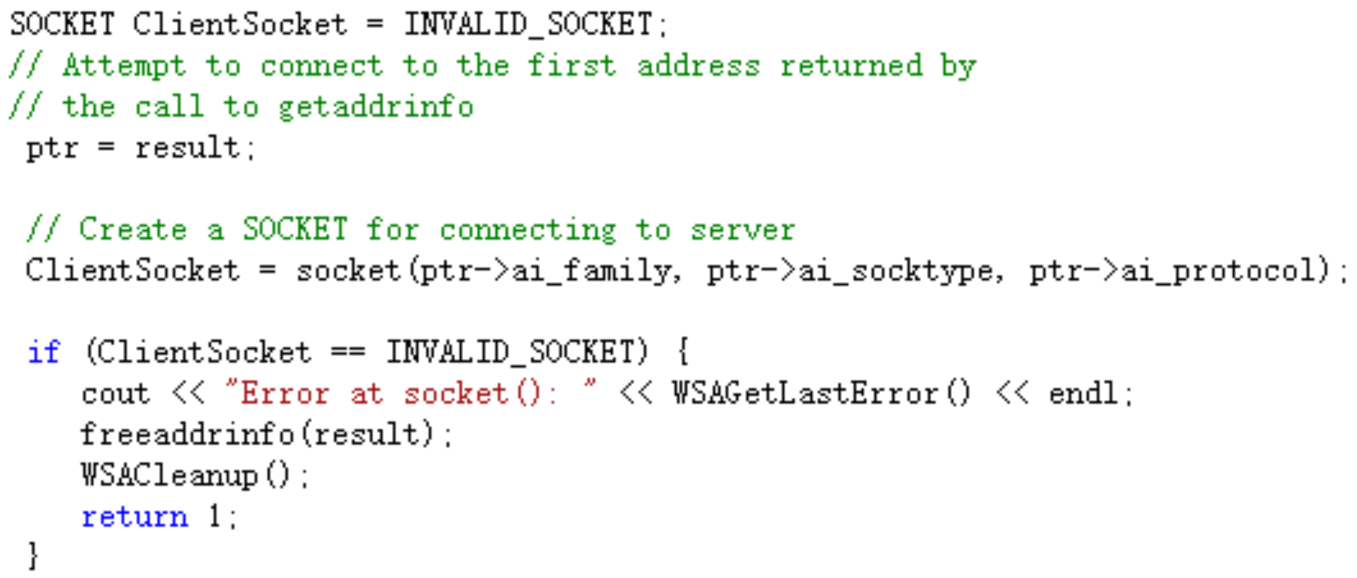


```
#define DEFAULT_PORT "80"
```



```
// Resolve the server address and port
iResult = getaddrinfo(argv[1], DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
    cout << "getaddrinfo failed: " << iResult << endl;
    WSACleanup();
    return 1;
}
```

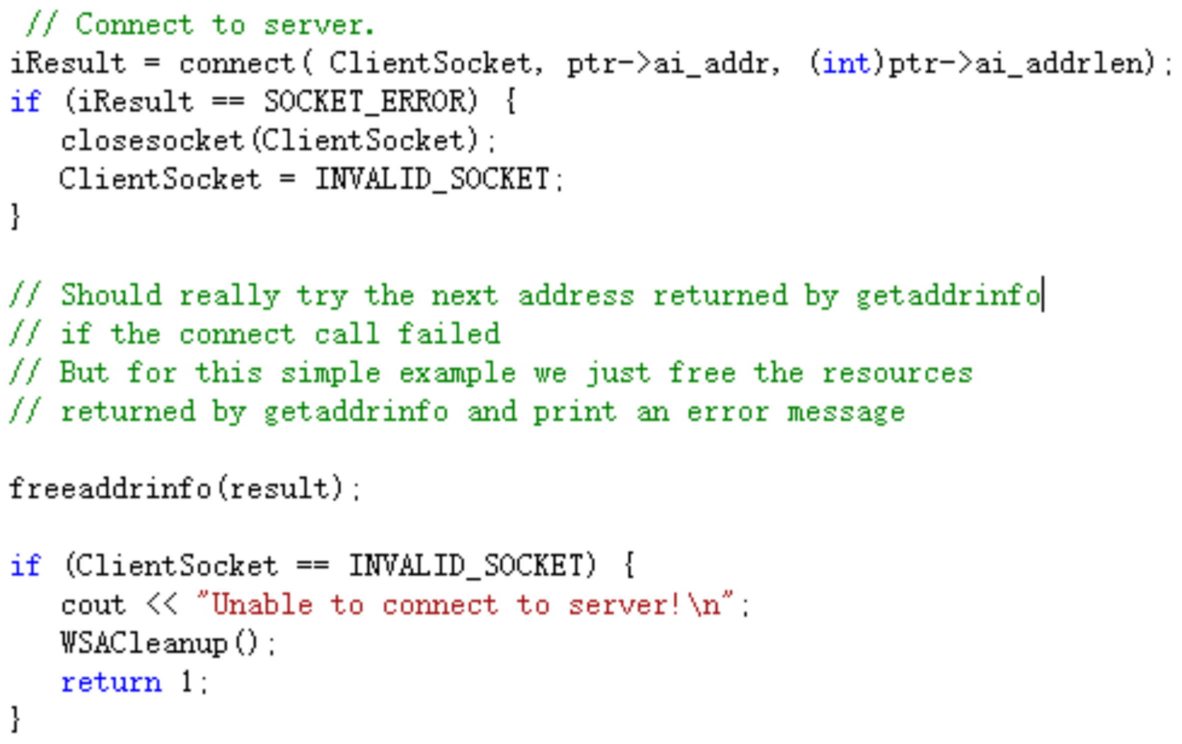
5. Create a SOCKET object called ClientSocket



```
SOCKET ClientSocket = INVALID_SOCKET;
// Attempt to connect to the first address returned by
// the call to getaddrinfo
 ptr = result;

 // Create a SOCKET for connecting to server
 ClientSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);

 if (ClientSocket == INVALID_SOCKET) {
    cout << "Error at socket(): " << WSAGetLastError() << endl;
    freeaddrinfo(result);
    WSACleanup();
    return 1;
 }
```

A simple WebClient

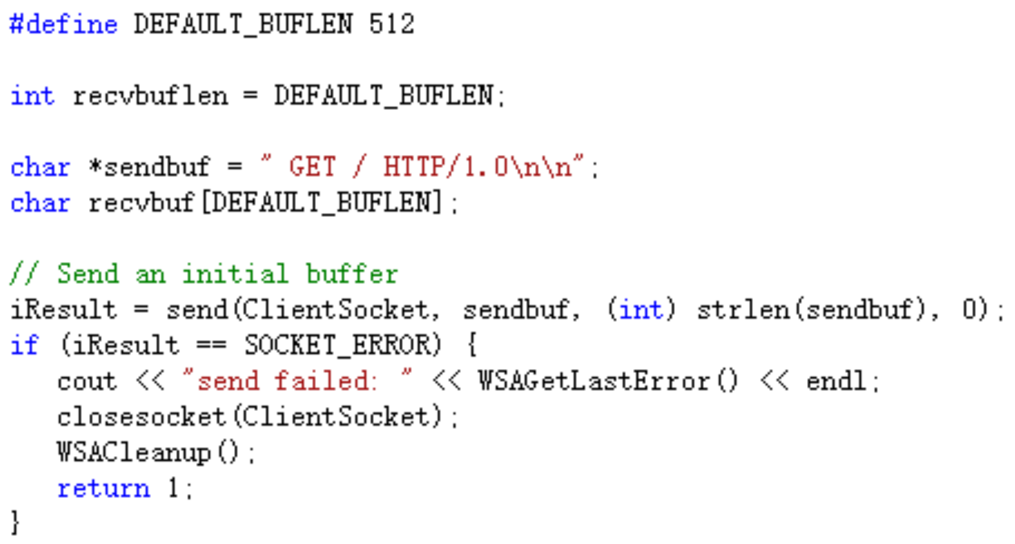
6. Call the connect function to connect to the Web server.



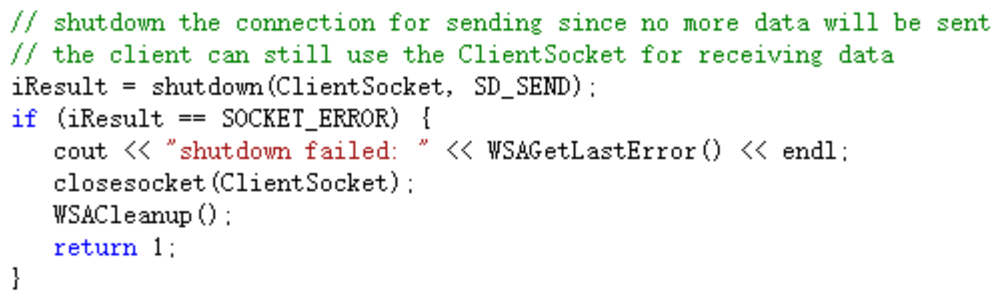
```
 // Connect to server.
iResult = connect( ClientSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    closesocket(ClientSocket);
    ClientSocket = INVALID_SOCKET;
}

// Should really try the next address returned by getaddrinfo
// if the connect call failed
// But for this simple example we just free the resources
// returned by getaddrinfo and print an error message

freeaddrinfo(result);

if (ClientSocket == INVALID_SOCKET) {
    cout << "Unable to connect to server!\n";
    WSACleanup();
    return 1;
}
```

A simple WebClient

7. Send http get and shutdown the sending side of the socket



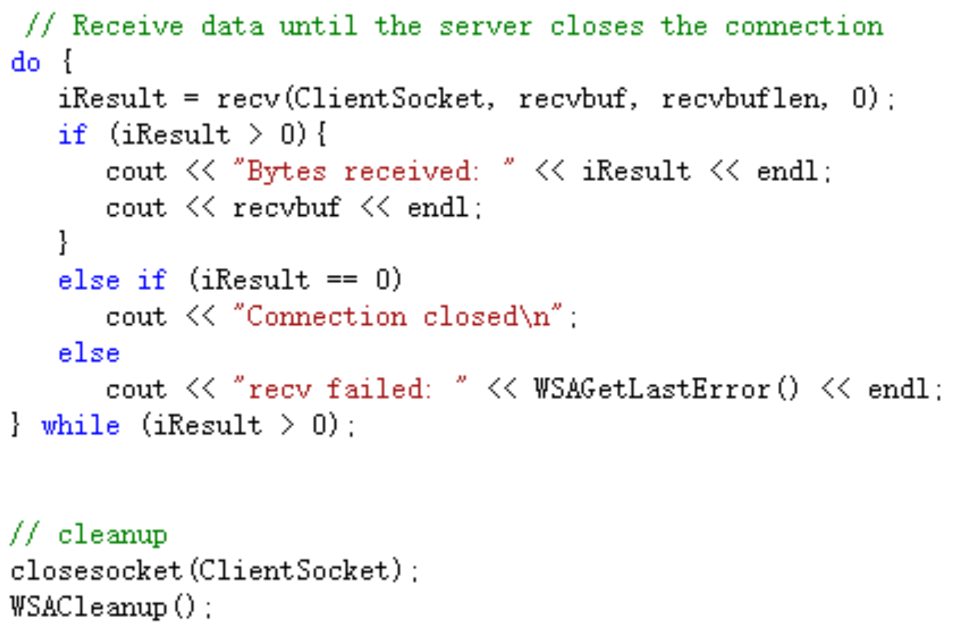
```
#define DEFAULT_BUFLEN 512

int recvbuflen = DEFAULT_BUFLEN;

char *sendbuf = " GET / HTTP/1.0\n\n";
char recvbuf[DEFAULT_BUFLEN];

// Send an initial buffer
iResult = send(ClientSocket, sendbuf, (int) strlen(sendbuf), 0);
if (iResult == SOCKET_ERROR) {
    cout << "send failed: " << WSAGetLastError() << endl;
    closesocket(ClientSocket);
    WSACleanup();
    return 1;
}

cout << "Bytes Sent: " << iResult << endl;
```



```
// shutdown the connection for sending since no more data will be sent
// the client can still use the ClientSocket for receiving data
iResult = shutdown(ClientSocket, SD_SEND);
if (iResult == SOCKET_ERROR) {
    cout << "shutdown failed: " << WSAGetLastError() << endl;
    closesocket(ClientSocket);
    WSACleanup();
    return 1;
}
```

A simple WebClient

8. Receive data from server and cleanup



```
 // Receive data until the server closes the connection
do {
   iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
   if (iResult > 0){
      cout << "Bytes received: " << iResult << endl;
      cout << recvbuf << endl;
   }
   else if (iResult == 0)
      cout << "Connection closed\n";
   else
      cout << "recv failed: " << WSAGetLastError() << endl;
} while (iResult > 0);


// cleanup
closesocket(ClientSocket);
WSACleanup();
```

Contents

A simple web server

- A TCP server
  - Accept a single connection from a web browser.
    - port 80
  - Responds with an HTML message.

A simple web server

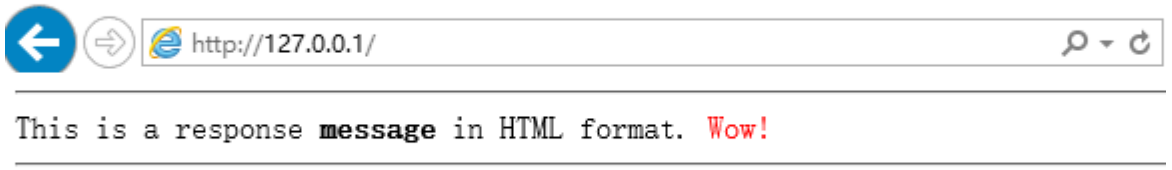
- Hints

```
#define DEFAULT_PORT "80"
char outbuf[DEFAULT_BUFLEN];

  // Receive from the Web browser
  //  iResult from recv() is the number of bytes received
  iResult = recv(ClientSocket, recvbuf, BUF_SIZE, 0);
  for (i=0; i<iResult; i++)
    printf ("%c", recvbuf[i]);

  // Copy the HTML response into the out buffer
  strcpy_s(outbuf, "<html><body><hr>This is a response <b>message</b> in HTML
format. <font color=red>Wow!</font><hr></body></html>");

  // Send HTML response to the client
  send(ClientSocket, outbuf, strlen(outbuf), 0);
```

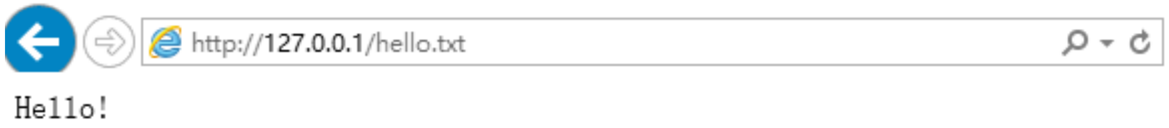
Contents

E4a: An Extended web server

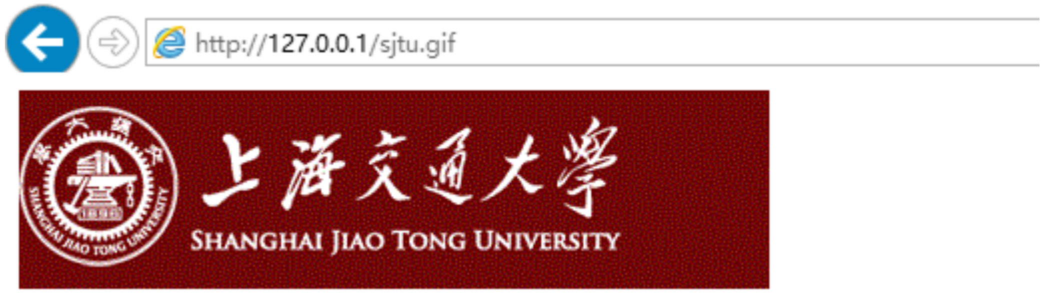
- Write an extended Web server for Windows
  - Handles multiple clients
  - Serves HTML, text, and GIF images


http://127.0.0.1/

This is a response **message** in HTML format. Wow!


http://127.0.0.1/hello.txt

Hello!


http://127.0.0.1/sjtu.gif

E4a: An Extended web server

```
#include <fcntl.h>          // For binary handle options
#include <sys\stat.h>      // For binary write()
#include <io.h>             // Needed for open(), close(), write()


//----- HTTP response messages ----------------------------------------------
#define OK_IMAGE  "HTTP/1.0 200 OK\r\nContent-Type:image/gif\r\n\r\n"
#define OK_TEXT   "HTTP/1.0 200 OK\r\nContent-Type:text/html\r\n\r\n"
#define NOTOK_404 "HTTP/1.0 404 Not Found\r\nContent-Type:text/html\r\n\r\n"
#define MESS_404  "<html><body><h1>FILE NOT FOUND</h1></body></html>"

//----- Defines -------------------------------------------------------------
#define  BUF_SIZE          1024     // Buffer size (big enough for a GET)
#define  PORT_NUM            80     // Port number for a Web server

//----- Function prototypes -------------------------------------------------
void handle_get(void *in_arg);       // Thread function to handle GET
```

```
// Main loop to listen, accept, and then spin-off a thread to handle the GET
 while(1)
 {
  printf("main loop: linstening ... \n");
  // Listen for connections and then accept
  listen(ListenSocket, 50);
  addr_len = sizeof(client_addr);
  client_s = accept(ListenSocket, (struct sockaddr *)&client_addr, &addr_len);
  if (client_s == -1)
  {
    printf("ERROR - Unable to create a socket \n");
    exit(1);
  }
  printf("client socket accepted, %d... \n",client_s);
  // Spin-off a thread to handle this request (pass only client_s)
  if (_beginthread(handle_get, 4096, (void *)client_s) < 0)
  {
    printf("ERROR - Unable to create a thread to handle the GET \n");
    exit(1);
  }
 }
 printf("main loop completed. close server socket... WSAcleanup \n");
 // Close the server socket and clean-up winsock
 closesocket(server_s);
 WSACleanup();
```

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