Preliminaries II

Dmitri Loguinov
Texas A&M University

August 30, 2022
Agenda

- Homework #1
  - HTTP basics
  - Windows sockets
HTTP Basics

• General URL format:
  - Optional elements shown in square brackets

```scheme://[user:pass@]host[:port][/path][?query][#fragment]```

• No need to parse username/password in this homework, but you have to strip off the fragment and extract the port number from the host
  - If the path is not present, must use root “/” in its place

• HTTP request is [/path][?query]

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Request /</td>
<td>Request /cs/in:dex.php?test=1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL: <a href="http://tamu.edu?test=1/blah">http://tamu.edu?test=1/blah</a></th>
<th>URL: <a href="http://tamu.edu?tes:t=1/blah">http://tamu.edu?tes:t=1/blah</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Request /?test=1/blah</td>
<td>Request /?tes:t=1/blah</td>
</tr>
</tbody>
</table>

1) Find # using strchr() and truncate
2) Find ?, extract query, truncate
3) Find /, extract path, truncate
4) Find :, extract port, truncate, obtain host
HTTP Basics 2

- HTTP request message
  - Begins with the method line, followed by (field: value) pairs
  - Ends with an empty line

- Methods in hw1
  - GET and HEAD, same syntax

- HTTP responses
  - Status line begins with HTTP/
  - Status codes are 3-digit integers

minimal request

GET /courses/ HTTP/1.0
Host: irl.cs.tamu.edu
Connection: close

HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html
Server: Microsoft-IIS/7.0
X-Powered-By: ASP.NET
MicrosoftOfficeWebServer: 5.0_Pub
MS-Author-Via: MS-FP/4.0
Date: Thu, 17 Jan 2013 09:22:34 GMT
Connection: close
Content-Length: 16367

<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
...
Agenda

• Homework #1
  - HTTP basics
  - Windows Sockets
Windows Sockets

• Sockets are interfaces to the TCP/IP protocol stack
  – More on TCP and IP later in the semester
  – HTTP (hw1) uses TCP
  – Sockets identified by their handle

• Communication using sockets is accomplished by a set of system calls to Winsock
  – Winsock is the Windows implementation of sockets
  – Parts are identical to Berkeley sockets on Unix

• TCP sockets can be used in two modes:
  – Client (socket actively establishes outgoing connections)
  – Server (socket listens for incoming connections)
Windows Sockets 2

- **IP address**: uniquely identifies the host to be contacted
  - 4-byte number written with a dot between each byte
  - How to assign IP 128.194.135.60 to an integer in C++?
- **Localhost** has IP address 127.0.0.1
- What if multiple network applications need to be run simultaneously on one host?
- **Solution**: ports
  - Each socket can be bound to a unique port
  - Socket = OS handle, port = externally visible identifier
  - The OS forwards incoming messages to sockets based on ports they are bound to
Windows Sockets 3

• Ports are 2-byte unsigned integers
  - Port 0 reserved, 1-1023 are system; 1024-65535 user
• Some well-known ports
  - HTTP: 80 (sometimes also 8000 or 8080)
  - Telnet: 23
  - SSH: 22
  - SMTP: 25 (encrypted SMTP on 465, 587)
• See http://www.iana.org/assignments/port-numbers
• When issuing a connect, the OS implicitly binds the socket to the next available port
  - Clients do not need to worry about their port numbers
  - But explicit binding is mandatory for servers
Example (Windows)

- Use “netstat –a” to see open ports on your host

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>viper:echo</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:discard</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:daytime</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:qotd</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:chargen</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:epmap</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:microsoft-ds</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:netbios-ssn</td>
<td>viper:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:3713</td>
<td>imap.cs.tamu.edu:pop3</td>
<td>TIME_WAIT</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:3717</td>
<td>google.com:http</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>viper:3718</td>
<td>google.com:http</td>
<td>ESTABLISHED</td>
</tr>
</tbody>
</table>

P2P application (skype, BitTorrent) or possibly hacker
Windows Sockets 4

- Winsock requires initialization (unlike Unix)
  - This should be done before any other winsock calls
  - Once per program execution

```c
#include <windows.h>
WSADATA wsaData;
WORD wVersionRequested = MAKEWORD(2,2);
if (WSAStartup(wVersionRequested, &wsaData) != 0)
{
    printf("WSAStartup error %d\n", WSAGetLastError());
    exit(-1);
}
```
Agenda

- HTTP basics
- Windows sockets
  - Clients
Clients

• **Steps to writing a TCP client:**
  - Open a socket
  - Determine the IP address of the server in URL
  - Initiate connection with the server
  - Send request
  - Receive response
  - Close socket

• **Task 1: open/close a TCP socket**
  - Sockets are initially unbound (i.e., no port associated)

```c
SOCKET sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock == INVALID_SOCKET) {
    printf("socket() error %d\n", WSAGetLastError());
    exit(-1);
}
...
closesocket (sock);
```
- **Task 2**: determine the IP address of the server in URL
- First assume the server is specified by an IP address
  - Try converting to 4-byte int using `inet_addr(host);`
- If this fails, then the server is given by its hostname
  - Use the domain name system (DNS)
  - DNS resolves **fully-qualified domain names** (FQDN) such as `www.tamu.edu` to their IP addresses (165.91.22.70)
- DNS lookup performed through a system call
  - `struct hostent* remote = gethostbyname(host);`
  - Returns 4-byte IP addresses inside the structure
• **Task 3**: connect socket to server on given port

```c
struct sockaddr_in server;
server.sin_family = AF_INET; // IPv4
server.sin_addr = ... // from inet_addr or gethostbyname
server.sin_port = ... // port #

if (connect (sock, (struct sockaddr*) &server,
             sizeof(struct sockaddr_in)) == SOCKET_ERROR)
{
    printf ("Connection error: %d\n", WSAGetLastError());
    return;
}
```

• **Main caveat** is that all numbers must be in network byte order (MSB first)
  - Forward (host-to-network): htons(), htonl()
  - Reverse (network-to-host): ntohs(), ntohl()

• `inet_addr` and `gethostbyname` internally perform this, so usually only port # needs explicit conversion
**Clients 4**

- **Task 4**: send request conforming to correct protocol

```c
char *sendBuf = new char [requestLength + 1];  // +1 for NULL
// place request into buf
if (send (sock, sendBuf, requestLen, 0) == SOCKET_ERROR)
{
    printf ("Send error: %d\n", WSAGetLastError());
    return;
}
```

- **Task 5**: receive response into recvBuf
  - Data arrives in chunks from function recv(), needs to be appended to a character buffer
  - Size of message and each chunk is unknown a-priori
    - Recv() must be called repeatedly until it returns 0 bytes
    - Use a pointer that moves along receive buffer
    - Buffer starts from 4-8 KB and is resized dynamically to accommodate longer messages