CSCE 463/612: Networks and Distributed Processing

Homework 1 Part 1 (25 pts)

Due date: 1/24/24

1. Purpose

This assignment builds an understanding of text-based application-layer protocols, multi-threading, system APIs, and Windows sockets.

2. Problem Description

Using Visual C++, your goal is to create a simple web client that accepts URLs and then crawls them to display basic server/page statistics.

2.1. Code (25 pts)

Your program must accept a single command-line argument with a target URL. If the argument is missing or there are too many of them, print usage information and quit. You should be able to handle URLs that fall under this general format

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

where the only acceptable scheme in this homework is "http". Examples (IRL servers require TAMU VPN):

```
hw1.exe http://tamu.edu                                      [no path, no port, no query]
hw1.exe http://www.tamu.edu:80                               [no path, port]
hw1.exe http://128.194.135.72:80/courses/index.asp#location [IP, port, path, fragment]
hw1.exe http://165.91.22.70/                                 [IP, path]
hw1.exe http://s2.irl.cs.tamu.edu/IRL7                       [64MB HTML file]
hw1.exe http://s2.irl.cs.tamu.edu/IRL8                       [128MB HTML file with zeros]
hw1.exe http://facebook.com:443?addrbook.php                 [no path, query, non-HTTP response]
hw1.exe http://relay.tamu.edu:465/index.html                 [path, port, recv error]
hw1.exe http://ftp.gnu.org:21/                               [recv timeout]
hw1.exe http://s22.irl.cs.tamu.edu:990/view?test=1           [path, port, query, no DNS]
hw1.exe http://128.194.135.25?viewcart.php/                  [no path, query, connect timeout]
```

Note that if the path is omitted, you must use the root directory / in its place. If the URL passes basic checks (i.e., correct scheme, non-zero port number), you should attempt to obtain the page via HTTP 1.0. Note that HTTP 1.1 allows the server to chunk the transfer, which is harder to decode (see the extra-credit section at the end of part 3). It is therefore important to request the version of HTTP that you can process.

If you manage to connect and receive a valid reply (HTTP 2xx), parse the HTML result and display the required information about your download (see below); otherwise, the program should legibly report the type of error encountered and terminate gracefully, *even if the remote host is hanging or not responding*. Note that your code must be able to handle pages of arbitrary length by dynamically expanding the buffer provided to `recv()`.

The following two examples show the required behavior during successful downloads:

```
URL: http://www.dmoz.org
        Parsing URL... host www.dmoz.org, port 80, request /
        Doing DNS... done in 655 ms, found 149.174.98.43
      * Connecting on page... done in 47 ms
        Loading... done in 141 ms with 17839 bytes
        Verifying header... status code 200
      + Parsing page... done in 0 ms with 102 links
---------------------------------------
HTTP/1.1 200 OK
Date: Sun, 18 Jan 2015 01:52:21 GMT
Server: Apache
Set-Cookie: JSESSIONID=2C016DC9413B184589C279886C0BA9D0; Path=/
Content-Length: 17621
Connection: close
Content-Type: text/html;charset=UTF-8
```

and

```
URL: http://128.194.135.72
        Parsing URL... host 128.194.135.72, port 80, request /
        Doing DNS... done in 0 ms, found 128.194.135.72
      * Connecting on page... done in 0 ms
        Loading... done in 78 ms with 6957 bytes
        Verifying header... status code 200
      + Parsing page... done in 0 ms with 10 links
---------------------------------------
HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 6633
Content-Type: text/html
Server: Microsoft-IIS/7.0
Set-Cookie: ASPSESSIONIDAASDCQDS=FIMPNKHBEFLNGPCOGOOPPBHI; path=/
X-Powered-By: ASP.NET
MicrosoftOfficeWebServer: 5.0_Pub
MS-Author-Via: MS-FP/4.0
Date: Sun, 18 Jan 2015 01:54:01 GMT
Connection: close
```

Note that one-tab indentation after the first line, an asterisk for the connection phase, a plus for the parsing phase, and timing of each networking step (e.g., using `clock()`) are required. The printout following a horizontal line contains only the HTTP header.

If you manage to receive the page, but the status code is not 2xx, skip the HTML parser, but print everything else:

```
URL: http://www.yahoo.com
        Parsing URL... host www.yahoo.com, port 80, request /
        Doing DNS... done in 484 ms, found 98.138.253.109
      * Connecting on page... done in 31 ms
        Loading... done in 109 ms with 1746 bytes
        Verifying header... status code 301
---------------------------------------
HTTP/1.0 301 Redirect
Date: Sun, 18 Jan 2015 01:39:06 GMT
Via: http/1.1 ir24.fp.ne1.yahoo.com (ApacheTrafficServer)
Server: ATS
Location: https://www.yahoo.com/
Content-Type: text/html
Content-Language: en
Cache-Control: no-store, no-cache
Connection: keep-alive
Content-Length: 1450
```

If you are unable to download the page, stop at the last attempted step and display the failure condition (for network errors, provide the corresponding `WSAGetLastError()` result):

```
URL: http://facebook.com:443?addrbook.php
        Parsing URL... host facebook.com, port 443, request /?addrbook.php
        Doing DNS... done in 5 ms, found 31.13.93.35
      * Connecting on page... done in 15 ms
        Loading... failed with non-HTTP header (does not begin with HTTP/)
```

```
URL: http://relay.tamu.edu:465/index.html
        Parsing URL... host relay.tamu.edu, port 465, request /index.html
        Doing DNS... done in 4 ms, found 148.163.139.245
      * Connecting on page... done in 2 ms
        Loading... failed with 10054 on recv
```

```
URL: http://128.194.135.25?viewcart.php/
        Parsing URL... host 128.194.135.25, port 80, request /?viewcart.php/
        Doing DNS... done in 4 ms, found 128.194.135.11
      * Connecting on page... failed with 10060
```

```
URL: http://s22.irl.cs.tamu.edu:990/view?test=1
        Parsing URL... host s22.irl.cs.tamu.edu, port 990, request /view?test=1
        Doing DNS... failed with 11001
```

```
URL: http://ftp.gnu.org:21
        Parsing URL... host ftp.gnu.org, port 21, request /
        Doing DNS... done in 96 ms, found 209.51.188.20
      * Connecting on page... done in 31 ms
        Loading... failed with timeout
```

```
URL: http://xyz.com:/
        Parsing URL... failed with invalid port
```

```
URL: http://xyz.com:0
        Parsing URL... failed with invalid port
```

```
URL: ftp://yahoo.com
        Parsing URL... failed with invalid scheme
```

2.2. General Guidelines

Efficient coding and well-structured programming is expected. You may lose points for copy-pasting the same function (with minor changes) over and over again, for writing poorly designed or convoluted code, *not checking for errors in every API you call*, and allowing buffer overflows, access violations, debug-assertion failures, heap corruption, synchronization bugs, memory leaks, or any conditions that lead to a crash. Furthermore, your program must be robust against unexpected responses from the Internet and deadlocks.

Basic operation of Winsock is covered in class, with supporting examples provided in the sample homework project on the course website. Additional caveats are discussed next.

2.3. Receive Loop

Reading from sockets is accomplished using this general algorithm that resizes the buffer as needed:

```
class Socket {
```

```
        SOCKET  sock;                  // socket handle
        char    *buf;                  // current buffer
        int     allocatedSize;         // bytes allocated for buf
        int     curPos;                // current position in buffer
        ...                            // extra stuff as needed
};

Socket::Socket ()
{
        // create this buffer once, then possibly reuse for multiple connections in Part 3
        buf = ...      // either new char [INITIAL_BUF_SIZE] or malloc (INITIAL_BUF_SIZE)
        allocatedSize = INITIAL_BUF_SIZE;
}

bool Socket::Read (void)
{
        // set timeout to 10 seconds
        while (true)
        {
                // wait to see if socket has any data (see MSDN)
                if ((ret = select (0, &fd, ..., timeout)) > 0)
                {
                        // new data available; now read the next segment
                        int bytes = recv (sock, buf + curPos, allocatedSize – curPos, ...);

                        if (errors)
                                // print WSAGetLastError()
                                break;

                        if (connection closed)
                                // NULL-terminate buffer
                                return true;           // normal completion

                        curPos += bytes;               // adjust where the next recv goes

                        if (allocatedSize – curPos < THRESHOLD)
                                // resize buffer; you can use realloc(), HeapReAlloc(), or
                                // memcpy the buffer into a bigger array
                }
                else if (timeout)
                        // report timeout
                        break;
                else
                        // print WSAGetLastError()
                        break;
        }

        return false;
}
```

The above fragment checks the socket to see if there is any data before attempting a receive. Without this, you may experience deadlocks inside `recv()` when the remote host neither provides any data nor closes the connection. Since `select()` modifies the parameters you pass to it, you must reinsert `sock` into `fd_set` each time you call `select()`. This is accomplished with macros FD_ZERO and FD_SET. For more details, see

http://msdn.microsoft.com/en-us/library/ms740141(VS.85).aspx

A cleaner alternative to traditional Unix-style `select()` is `WSAEventSelect()` or the IOCP framework. The former lets you register an event that gets signaled when the socket has data in it. This allows your code to wait for multiple events and implement simple timeout-based socket disconnection. The latter is much more complicated and should be attempted only if the rest of the homework appears too simple:

http://msdn.microsoft.com/en-us/library/windows/desktop/aa365198(v=vs.85).aspx

2.4. Required HTTP Fields

The format of GET requests was shown in class. At minimum, you need to transmit the request line and the host string with the name of the server. For example:

```
GET /some/page/index.php?status=15 HTTP/1.0
Host: tamu.edu
```

However, this request may keep the connection open for some non-compliant servers, which makes it difficult to detect the end of transfer. You therefore may want to explicitly request that the server close the connection:

```
GET /some/page/index.php?status=15 HTTP/1.0
Host: tamu.edu
Connection: close
```

It is also common courtesy to specify your user-agent to keep webmasters aware of visiting browsers and robots. In fact, some websites (e.g., akamai.com) refuse to provide a response unless the user-agent is present in the request header:

```
GET /some/page/index.php?status=15 HTTP/1.0
User-agent: myTAMUcrawler/1.0
Host: tamu.edu
Connection: close
```

You should invent your own string in the format of `crawlerName/x.y`, where x.y can evolve from 1.1 to 1.3 as you progress through the parts of this homework.

2.5. Parser

The sample parser solution (from the course website) contains four library (.lib) files, which need to be copied into your project's folder with .cpp files. *Do not add lib files into the project in Visual Studio.* You additionally need HTMLParserBase.h, which should be included into precompiled headers (`pch.h` or `stdafx.h`). There is also no need to add .lib files into linker input since `HTMLParserBase.h` already does this using #pragma directives. There are four different libraries that cover all possible combinations of Debug/Release/win32/x64, where the proper file is automatically determined by `HTMLParserBase.h`.

You may also run into an issue with Release mode when the newest Visual Studio refuses to include HTML parser libraries because they're from an earlier version of the compiler. The solution is to disable whole program optimization (Project Properties → C/C++ → Optimization).

2.6. Helpful Functions, Tools, and Commands

You can use C-string functions `strchr` and `strstr` to quickly find substrings in a buffer. Comparison is usually performed using `strcmp/stricmp` or `strncmp/strnicmp`. It is recommended to use `printf` as it greatly reduces the amount of typing in this homework compared to `cout`. You can also use `sprintf` to assemble the various parts of a request.

Oftentimes, it is convenient to declare a fixed-size buffer that is large enough to accept even the longest link. To help with this, `HTMLParserBase.h` defines two constants `MAX_HOST_LEN` and `MAX_REQUEST_LEN` that upper-bound the examples we consider valid for this homework. If the user provides a string that violates either bound, you should reject it. An explicit check is required, especially in Part 3 where some of the crawled URLs are known to violate the maximum allowed host length.

Usage of `gethostbyname` for DNS lookups, printout of IPs via `inet_ntoa`, and connection to a server are provided in the sample solution.

For debugging responses, use an HTTP sniffer, e.g., http://websniffer.com, http://testuri.org/sniffer, or Firefox add-ons. If you need to see the contents of your outgoing packets, use http://www.wireshark.org/. For information about your network configuration, run `ipconfig` at the command prompt (to see the DNS servers, use `ipconfig /all`). To manually perform DNS lookups, try `nslookup host` or `nslookup IP`.

463/612 Homework 1 Grade Sheet (Part 1)

Name: _____

Function	Points	Break down	Item	Deduction
Input	1	1	No usage info if incorrect arguments	
Request	3	1	Incorrect GET syntax	
		1	No hostname in request	
		1	No user-agent in request	
Receive loop	4	1	No dynamic buffer resizing	
		2	Fails to receive/parse large files	
		1	No select()	
Output	10	3	Incorrect host/port/request	
		1	Incorrect DNS info	
		1	No timing of connect()	
		1	No timing of recv()	
		1	Incorrect page size	
		1	Incorrect HTTP status	
		1	Incorrect number of links	
		1	Incorrect HTTP header shown	
Errors	6	1	Does not handle invalid port/scheme	
		1	Does not notify of DNS failure	
		1	Does not notify of connect failure	
		1	Does not notify of recv timeout/failure	
		1	Does not notify of non-HTTP reply	
		1	Parses non-2xx pages	
Other	1	1	Missing files for compilation	

Additional deductions are possible for memory leaks and crashing.

Total points: _____