CSCE 463/612: Networks and Distributed Processing

Homework 1 Part 2 (25 pts)

Due date: 9/5/23

1. Problem Description

You will now expand into downloading multiple URLs from an input file using a single thread. To ensure politeness, you will hit only unique IPs and check the existence of `robots.txt`. Robot exclusion is an informal standard that allows webmasters to specify which directories/files on the server are prohibited from download by non-human agents. See http://www.robotstxt.org/ and https://en.wikipedia.org/wiki/Robots_exclusion_standard for more details. To avoid hanging the code on slow downloads, you will also have to abort all pages that take longer than 10 seconds[1] or occupy more than 2 MB (for robots, this limit is 16 KB).

1.1. Code (25 pts)

The program must now accept either one or two arguments. In the former case, it implements the previous functionality; in the latter case, the first argument indicates the number of threads to run and the second one the input file:

```
hw1.exe 1 URL-input.txt
```

If the number of threads does not equal one, you should reject the parameters and report usage information to the user. Similarly, if the file does not exist or cannot be successfully read, the program should complain and quit. Assuming these checks pass, you should load the file into RAM and split it into individual URLs (one line per URL). You can use `fopen`, `fgets`, `fclose` (or their `ifstream` equivalents) to scan the file one-line-at-a-time. A faster approach is load the entire input into some buffer and then separately determine where each line ends. Use C-style `fread` or an even-faster `ReadFile` for this purpose (the sample HTML-parser project shows usage of `ReadFile`). In the former case, note that the file must be opened in binary mode (e.g., using "rb" in `fopen`) to avoid unnecessary translation that may corrupt the URLs.

To avoid redundant DNS lookups, make sure that only unique hosts make it to `gethostbyname`. Combining this with an earlier discussion of politeness and robots leads to the following logic:

Parse URL → Check host is unique → DNS lookup → Check IP is unique → Request robots → Check HTTP code → Request page → Check HTTP code → Parse page

Note that robot existence should be verified using a HEAD request. This ensures that you receive only the header rather than an entire file. Codes 4xx indicate that the robot file does not exists and the website allows unrestricted crawling. Any other code should be interpreted as preventing further contact with that host. Your printouts should begin with indication that you read the file and its size, followed by the following trace:

[1] Your previous usage of `select` constrained each `recv` to 10 seconds, but allowed unbounded delays across the page. In these cases, a website feeding one byte every 9 seconds could drag forever.

```
Opened URL-input.txt with size 66152005
URL: http://www.symantec.com/verisign/ssl-certificates
        Parsing URL... host www.symantec.com, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 139 ms, found 104.69.239.70
        Checking IP uniqueness... passed
        Connecting on robots... done in 5 ms
        Loading... done in 57 ms with 213 bytes
        Verifying header... status code 200
URL: http://www.weatherline.net/
        Parsing URL... host www.weatherline.net, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 70 ms, found 216.139.219.73
        Checking IP uniqueness... passed
        Connecting on robots... done in 11 ms
        Loading... done in 61 ms with 179 bytes
        Verifying header... status code 404
      * Connecting on page... done in 3020 ms
        Loading... done in 87 ms with 10177 bytes
        Verifying header... status code 200
      + Parsing page... done in 0 ms with 16 links
URL: http://abonnement.lesechos.fr/faq/
        Parsing URL... host abonnement.lesechos.fr, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 1 ms, found 212.95.72.31
        Checking IP uniqueness... passed
        Connecting on robots... done in 138 ms
        Loading... done in 484 ms with 469 bytes
        Verifying header... status code 404
      * Connecting on page... done in 4335 ms
        Loading... done in 899 ms with 57273 bytes
        Verifying header... status code 200
      + Parsing page... done in 1 ms with 63 links
```

Note that you no longer need to print the request after the port. Uniqueness-verification steps and the robot phase are new and highlighted in bold. If you already have a function that connects to a server, downloads a given URL, and verifies the HTTP header, you can simply call it twice to produce both robots and page-related statistics. The function needs to accept additional parameters that specify a) the HTTP method (i.e., HEAD or GET); b) valid HTTP codes (i.e., 2xx for pages, 4xx for robots); c) maximum download size (i.e., 2 MB for pages, 16 KB for robots); and d) presence of an asterisk in the output. If any of the steps fails, you should drop the current URL and move on to the next:

```
URL: http://allafrica.com/stories/201501021178.html
        Parsing URL... host allafrica.com, port 80
        Checking host uniqueness... failed
URL: http://architectureandmorality.blogspot.com/
        Parsing URL... host architectureandmorality.blogspot.com, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 19 ms, found 216.58.218.193
        Checking IP uniqueness... failed
URL: http://aviation.blogactiv.eu/
        Parsing URL... host aviation.blogactiv.eu, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 218 ms, found 178.33.84.148
        Checking IP uniqueness... passed
        Connecting on robots... done in 9118 ms
        Loading... failed with 10060 on recv
URL: http://zjk.focus.cn/
        Parsing URL... host zjk.focus.cn, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 1135 ms, found 101.227.172.52
        Checking IP uniqueness... passed
        Connecting on robots... done in 367 ms
        Loading... done in 767 ms with 140 bytes
```

```
        Verifying header... status code 403
      * Connecting on page... done in 3376 ms
        Loading... failed with slow download
URL: http://azlist.about.com/a.htm
        Parsing URL... host azlist.about.com, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 81 ms, found 207.126.123.20
        Checking IP uniqueness... passed
        Connecting on robots... done in 5 ms
        Loading... failed with exceeding max
URL: http://apoyanocastigues.mx/
        Parsing URL... host apoyanocastigues.mx, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 57 ms, found 23.23.109.126
        Checking IP uniqueness... passed
        Connecting on robots... done in 49 ms
        Loading... done in 2131 ms with 176 bytes
        Verifying header... status code 404
      * Connecting on page... done in 3051 ms
        Loading... failed with exceeding max
URL: http://ba.voanews.com/media/video/2563280.html
        Parsing URL... host ba.voanews.com, port 80
        Checking host uniqueness... passed
        Doing DNS... done in 11 ms, found 128.194.178.217
        Checking IP uniqueness... passed
        Connecting on robots... done in 2 ms
        Loading... done in 490 ms with 2436 bytes
        Verifying header... status code 404
      * Connecting on page... done in 3001 ms
        Loading... done in 50 ms with 2850 bytes
        Verifying header... status code 408
```

In the last example, the downloaded page does not result in success codes `2xx`, which explains why parsing was not performed. As the text may scroll down pretty fast, you can watch for * and + to easily track how often the program attempts to load the target page and parse HTML, respectively.

1.2. Uniqueness

To maintain previously seen hosts and IPs, you can use the following verification logic:

```
#include <set>
#include <string>
using namespace std;

//---------
DWORD IP = inet_addr ("128.194.135.72");
set<DWORD> seenIPs;
seenIPs.insert(IP);
...
//---------
set<string> seenHosts;

// populate with some initial elements
seenHosts.insert("www.google.com");
seenHosts.insert("www.tamu.edu");

string test = "www.cse.tamu.edu";
auto result = seenHosts.insert (test);

if (result.second == true)
       // unique host
else
       // duplicate host
```

1.3. Page Buffers

Make sure to continue using a dynamic-buffering scheme for received pages in `Socket::Read`. Even though the maximum page size is now fixed, do not hardwire 2-MB buffers into your receiver. When you scale this program to 5000 threads in Part 3, such inefficient RAM usage may become problematic. Similarly, when you reuse the `Socket` class for the next connection, delete the old buffer if it is larger than 32 KB and start again with `INITIAL_BUF_SIZE`.

463/612 Homework 1 Grade Sheet (Part 2)

Name: _____

Function	Points	Break down	Item	Deduction
Output	11	1	Fails to show input file size	
		1	Incorrect URLs being crawled	
		1	Incorrect DNS results	
		1	Fails to print host checks	
		1	Fails to print IP checks	
		1	No timing of robots connect()	
		1	No timing of robots recv()	
		1	Incorrect robots page size	
		1	Incorrect robots HTTP status	
		2	Incorrect page download results	
Logic	8	2	Fails to load multiple pages	
		2	Allows duplicate hosts	
		2	Allows duplicate IPs	
		2	Loads robots-prohibited pages	
Robot errors	5	1	Does not notify of connect failure	
		1	Does not notify of recv failure	
		1	Does not notify of non-HTTP reply	
		1	Fails to report slow download	
		1	Fails to report exceeding max	
Other	1	1	Missing files for compilation	

Additional deductions are possible for memory leaks and crashing.

Total points: _____