CSE 3100 Systems Programming
Lab #7

Welcome to Lab 7! At the start of each lab, you will receive a document like this detailing the tasks you are to complete. Read it carefully. Try your best to finish labs during the lab session. However, labs are not due right after each lab period.

1 Get that web page and clean it

Prelude: Run the command `sudo make prereqs` to install the required software in your VM (`NGINX` and `html2text`). Then start the nginx server by typing (in a terminal) the command:

```
sudo nginx
```

The web is a complex place full of HTML that is barely readable. In this lab, your task is to produce two C programs (named `wgethtml` and `wgettext`) that retrieve and display an HTML document raw (`wgethtml`) and after stripping all the markup and only leaving pristine good-looking text (`wgettext`).

Namely, executing:

```
./wgettext http://localhost:80/index.nginx-debian.html
```

should retrieve (in this example) the welcome page of the NGINX server running in your VM and display its cleaned up content

```
****** Welcome to nginx! ******
If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.
```

On its standard output. As you might already know, this implies contacting the nginx WEB server at host `localhost` and on port number 80 to retrieve a named document (`index.nginx-debian.html`). In the example, the argument to wgettext is the URL (Universal Resource Locator) `http://localhost:80/index.nginx-debian.html` and it contains a few parts:

http://	This is the protocol being used. This part is ***optional.***
localhost	This is the name of the server to reach (to be resolved with `gethostbyname` of course). This part is ***mandatory***
:80	This is the port number to use to establish a connection to the named server. This part is ***optional*** and if omitted, your program should assume that the port is 80.
`/index.nginx-debian.html`	This is the name of the document to retrieve. This part is ***mandatory***

It should be clear that the following examples are all valid URLs:

```
http://localhost:80/index.nginx-debian.html
localhost:80/index.nginx-debian.html
http://localhost/index.nginx-debian.html
localhost/index.nginx-debian.html
```

At a minimum, the URL must specify the hostname and the document name. To successfully complete the lab, you have three tasks, listed below. Note that all 3 parts can be tested "standalone", but they build upon one another (complete them in order!)

Part 1: Parse that URL! (20pts)

Open the file `analyze.c` and write a C function

```
void analyzeURL(char* url,char* host,int* port,char* doc);
```

That analyzes the given URL (`url`) and breaks down the URL in its 3 components. Namely, it skips the protocol if present (always assumed to be **http://**), places in `host` a copy of the hostname, places in `port` the integer value of the port *(if specified)*, otherwise leaves port untouched and stores in `doc` a copy of the document name. For instance, on the example

```
http://localhost:80/index.nginx-debian.html
```

host would hold a copy of the string: "`localhost`"
port would contain the value `80`
`doc` would hold a copy of the string "`index.nginx-debian.html`"

Note that host and doc are fixed sized arrays allocated by the caller of analyzeURL and are large enough to accommodate long names (each should be 512 bytes). You do not need to write error recovery logic in case the names are longer.

Part 2: Connect the server... (40pts)

The second part is the central piece. You are to complete the implementation of `wgethtml.c`, the main program. The binary will take a URL on its command line, analyze it with the handy dandy function from part 1 and then connect to the WEB server specified via a TCP socket on the specified port. Thankfully web servers implement **very simple APIs** to retrieve HTML documents. To retrieve a document, a client establishes a TCP connection (like we have done many times) and sends a simple piece of text asking to "get" a document. Namely, the client sends the string:

```
GET /documentName\n
```

Then the server responds by opening a file that carries this name and sending the entire file, as plain text, back to the client over the same socket!

For instance, for the URL `http://localhost:80/index.nginx-debian.html`, your program would connect to `localhost` on port 80 issue the string (note the leading slash /):

```
GET /index.nginx-debian.html\n
```

Funnily enough, you could test this by installing telnet (`sudo apt-get install -y telnet`) and connecting to port 80 of `localhost` and issuing the command manually. The nginx WEB server responding at this port will then obediently dump the requested document into the socket for your usage. Now that the socket is brimming with data, use the supplied `readResponse` function to collect that HTML document and dump it on its standard output ***as is.***

Testing

To test your program, simply use:

```
./wgethtml http://localhost:80/index.nginx-debian.html
```

And you should have, as an output:

```
[localhost] [80] [/index.nginx-debian.html]
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed
and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Namely, the HTML served up by NGINX (and a line of printout at the start with the result of analyze).

Part 3: Query, retrieve, cleanup and display. (40pts)

While this may be a load of fun to read HTML, it's not very practical for us mere mortals. In part 3, you are expected to start by copying your solution to part 2 (`wgethtml.c`) into a new C source file `wgettext.c`. That is, your starting point is a program that happily retrieves HTML from a WEB server. Part 3 is all about *cleaning up that HTML*. Namely, we wish to strip all traces of HTML tags and only keep the text.

For instance, if the HTML payload from the server is:

```
<html>
<title>Hello world!</title>
<body>
    <p>The end of the tunnel is near!</p>
</body>
</html>
```

The output of the wgettext should be:

```
The end of the tunnel is near!
```

Thankfully, this is a functionality that already exists. Indeed, when you ran `make prereqs`, you installed a program `html2text` that does exactly that! It takes HTML **on its standard input and spits out text on its standard output**.

Your task is therefore to replicate this behavior and upgrade your program so that its output is plain text rather than HTML!

Testing

Once again, to test, simply run the command:

```
./wgettext http://localhost:80/index.nginx-debian.html
```

And you should be greeted by the textual version of the HTML served by NGINX which you could also see by running the output of wgethtml into text2html. Namely:

```
[localhost] [80] [/index.nginx-debian.html]
****** Welcome to nginx! ******
If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.
Thank you for using nginx.
```