

EECS 498 W26 System Design of a Search Engine

08 LinuxGetUrl

Due Sunday, February 15, 2026, 11:59 pm

This is the first of two back-to-back assignments intended to help you understand how to read webpages.

In this assignment, you will read webpages using unencrypted HTTP protocol by completing the code in a simple `LinuxGetUrl` utility.

Starter files

You are given these starter files in `LinuxGetUrlSslStarter.zip` for both this and the next assignment.

<code>LinuxGetUrl.cpp</code>	Starter file for the <code>LinuxGetUrl</code> utility to read webpages with unencrypted HTTP.
<code>LinuxGetSsl.cpp</code>	Starter file for the <code>LinuxGetSsl</code> utility you'll build in the next assignment to read webpages with encrypted HTTPS.
<code>Makefile</code>	A makefile for the <code>LinuxGetUrl</code> and <code>LinuxGetSsl</code> utilities.

The assignment

You are to complete the code in `LinuxGetUrl.cpp`. Here are the steps.

1. Parse the URL to find the host

This step is already implemented for you, in the `ParsedUrl` class. You can parse a URL with:

```
ParsedUrl url( argv[ 1 ] );
```

C-Strings `Service`, `Host`, `Port`, and `Path` will be parsed from the URL for you to use.

2. Find address details of host

Next get the address of the host that you are trying to connect to. Given the host name, port, and hints, you can retrieve the address with `getaddrinfo( )`. Hints passed into `getaddrinfo` as params helps narrow down the returned addresses. The addresses are returned as a linked list.

```
struct addrinfo *address, hints;
memset( &hints, 0, sizeof( hints ) );
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
getaddrinfo( url.Host, "80", &hints, &address );
```

3. Create a socket

Next, create a socket, an "endpoint" to communicate over the network, specifying a family (`AF_INET` is used to communicate over IPv4), a type (`SOCK_STREAM` is used to have TCP packets), and a protocol (`IPPROTO_TCP` is the default protocol for TCP packets).

You can either use the information from our hints struct from above, or you can hard code the information.

```
int socketFD = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
```

4. Connect to the server

To connect to a server, you need a socket file descriptor to connect with, an address to connect to, and the length of an address struct.

```
int connectResult = connect( socketFD, address->ai_addr, address->ai_addrlen );
```

5. Send a HTTP request

Next, send an HTTP request to the server you are trying to communicate with. Here is an example GET message.

```
GET / HTTP/1.1
Host: www.nytimes.com
User-Agent: LinuxGetUrl/2.0 nham@umich.edu (Linux)
Accept: */*
Accept-Encoding: identity
Connection: close
```

All the lines in a GET message must end in '\r\n', not just '\n'. Additionally, there must be an extra blank line at the end of the request, i.e., "Connection: close\r\n\r\n".

Some sites will not even respond without a User-Agent field. It is a free text field and can be anything if it exists. It is typically the name of the software product that generated the GET request, plus a slash followed by a version number. The OS or build environment is usually given in parentheses.

Pick a different name for your User-Agent and give YOUR contact info, not mine, so that complaints go to you, not me!

The Accept: and Accept-Encoding: fields are not required but are typically provided.

To send the GET request to the server, use the send() function.

```
send( socketFD, getMessage.c_str( ), getMessage.length( ), 0 );
```

6. Save the response

After sending the GET request, read the response from the server into a buffer. As long as the server is sending information, keep receiving the data. To accomplish this, use recv() in a while loop, exiting when there's no more data.

```
char buffer[ 10240 ];
int bytes;
while ( ( bytes = recv( socketFD, buffer, sizeof( buffer ), 0 ) ) > 0 )
    write( 1, buffer, bytes );
```

7. Cleaning up

Finally, clean up the resources, closing the socket and freeing the addrinfo.

```
close( socketFD );
freeaddrinfo( address );
```

You now have a fully functional HTTP webpage reader!

To make it suitable for autograding, your LinuxGetUrl and LinuxGetSsl programs should only write the returned file to stdout, not anything else.

The versions shown in class are instrumented to print out all the traffic, including the GET message and the server's response header, in addition to the actual content of the page. I did that to make it easier to demo in class and I forgot to make that clear. But all of that GET request and response header stuff will be different for you than for me. Any

timestamps will be different and your version will have a different User-agent name and email address. So, there's no point in comparing that.

The only part that should be the same for all of us is the file we get back, which starts right after the blank line at the end of the server's response header. So, that's the only thing I can test, and that's the only part you should write to stdout.

Autograder

Submit your `LinuxGetUrl.cpp` file to the autograder at <https://autograder.io/web/project/3489>.