PROJECT 4 - PSOOGLE
*Computer Systems Principles*

Emery Berger Mark Corner

November 23, 2010

# 1 Overview

Goals of this assignment: understanding networking and client server systems.

In this assignment, you will write a simple http web server that handles conjunctive search queries (like Google - searching for documents that match all of the query terms) and returns a list of hits.

# 2 Server

This server builds on the results of the first and second projects. Generally, your server will spider a set of web pages, build an inverted index of the pages, accept requests for *conjunctive searches*, and return results.

## 2.1 Input

Your program, to be called **server**, will take four command-line inputs:

1. The root URL to start from;
2. The maximum depth to crawl;
3. The number of worker threads to spawn;
4. The port number your server will run on.

(1)-(3) are exactly as in the **spider** project.

(4) is the port number the server will run on. Pick a large random number, such as the last 4 digits of your student id, to make sure you don't collide with someone else.

### 2.2 Output

Your program should print the inverted map to the screen (using a format similar to the **inverter** project, but you will print out the url values instead of document IDs). You can print the inverted map by modifying the print method. This output shows that you are spidering and inverting correctly. Apart from being useful for you in debugging your code, getting your code working until this stage properly accounts for 20% of the grade in this project.

Your program should then produce another line of output to the console: "indexing complete" and then start the server.

The rest of the interaction will take place through the web server itself.

The user will ask the server for one of two things, a file or a query. If it asks for a file (either index.html or the image file), the server will return it to the client. All other requests for files should cause the server to terminate the connection. In response to queries, the web server will return a set of web pages that contain all of the search terms.

Please use this code snippet to output the search results:

```
sprintf (buf2, "<a␣href=\"http://%s/%s\">%s␣%s</a></br>",
  host.c_str(), file.c_str(), host.c_str(), file.c_str());
```

## 3 Spidering and Indexing

All of the spidering is done exactly as described in the previous projects. Only grab the first `MAX_READ` bytes of each page as before, and only build an index from the words as described in the **inverter** project.

## 4 Interacting Over the Network

See simplesocket.h for the interface to the networking components.

The server should instantiate a new socket to accept requests from:

```
serversocket sock (port);
```

This completes all of the necessary binding and listening. After setting up the socket, your server can accept new requests from clients:

```
new_sock = sock.accept();
```

Note that `accept` returns a pointer to a new simplesocket. This is the socket you will use to communicate with this newly accepted client. Remember that eventually, you will need to close and delete this socket:

```
new_sock->close();
delete new_sock;
```

After accepting a connection, you can spawn a new thread to receive messages from this client:

```
HttpMessage http_msg = new_sock->ReceiveHttpMessage();
```

The type of message is contained in `http_msg.MessageType`, but you will only be concerned with GET messages; ignore everything else.

The message itself is parsed into `http_msg.Resource`.

Parse the HTTP request into either (1) a GET query for "/" or "/index.htm" whereupon you will read in the provided index.htm file from disk, send it over to the client and close the connection (this is fairly important) or (2) a GET query that encapsulates a search query. Search queries look a lot like GET queries. When conjunctively searching for the query "hello seeking world" this is how the requests look:

```
GET /?word1=hello&word2=seeking&word3=world HTTP/1.1
```

`/?` will precede every search query. The '&' delimits every query besides the first.

# 5 Compiling, Testing and Hints

## 5.1 Compiling

Use the following command to compile your server:

```
g++ -Wall -g -o server server.cc libspider.a -lpthread
```

Note that you need to copy libspider.a, url.h, and simplesocket.h to your current working directory (you should already have these files from the **spider** project).

Create a client that sends GET messages to your server. You can do one of two things:

1. Create a html page with forms, a submit button, and some javascript to submit your GET message to a hostname and portnumber. This will require you to open a web browser on the edlab machines and you may have issues from non-CS networks.

2. Use TELNET to send short http messages to your client. You can use TELNET to connect to your server's hostname and portnumber. Use command 'hostname' on your server machine if you are not sure what the hostname is. The following command connects to port number 2387 on hostname elnux2.cs.umass.edu: `telnet elnux2.cs.umass.edu 2387`. You will then get a telnet prompt: `telnet>`.

   You can send a get query by typing

   ```
   telnet>GET /?Word1=hello&word2=world HTML/1.0 <enter> <enter>
   ```

   Now telnet has sent the above GET message to your server. You can thus debug your server to ensure it handles different messages properly.

### 5.2 Testing and Grading

The code that will be used to test your server will send queries in the following format:

`GET /?word1=hello&word2=seeking&word3=world HTTP/1.1`

(the query terms are ”hello”, ”seeking” and ”world”).

Note that the following is a valid query message:

`GET /?word1=hello&word2=&word3=world HTTP/1.1`.

This query means that the second and third query terms are blank. In general, your code should be able to handle queries of 1 term, 2 terms, or 3 terms. The code that will be used to test your server will send file requests in the format `GET absolute_path_of_file_on_the_server/filename HTTP/1.1`.

Make sure your code is able to parse the above message formats to read the query words and filename arguments properly.

## 6 Handing Project In

For now, you will be handing in your project manually. Grading will be similar to the autograder, so if it doesn't compile, follow the input spec exactly, or produce the right input, it will be a 0.

Please mail your program (just server.cc) by the due date.

### 6.1 Email submission policy

- Please name your server C++ code `server_edlabID.cc`. For example, if your edlab ID is joel, your server file should be called `server_joel.cc`.
- Please email your solution only *once*, only when you are sure of your solution/
- We will not consider second submissions because these submissions are by email. Once a solution has been received, it will be treated as *final*.