

# CSE 123b – Spring 2004

## Programming Project #2 (version 1.0)

**Due: Friday, 6/4 by midnight**

Answers to any frequently asked questions that emerge will be available at  
<http://www.cs.ucsd.edu/classes/sp04/cse123b/Project2.FAQ>

---

## Overview

In this programming assignment, you will implement the part of a simple Gnutella 0.4 file-sharing service. Recall that Gnutella provides a distributed peer-to-peer search service – queries for content are propagated via flooding through the Gnutella overlay network and peers matching the search return responses indicating that they hold the content. Finally, matching content is downloaded by the requestor directly via the HTTP protocol. The full version of the 0.4 version of the Gnutella protocol is documented at: [http://www9.limewire.com/developer/gnutella\\_protocol\\_0.4.pdf](http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf) (note that this document uses the term *servent* in place of client or server – meant to reflect the fact that Gnutella nodes are potentially both clients and servers).

In this assignment, you will implement the client functions of Gnutella:

- 1) **Connecting** to the Gnutella network (given an initial node to connect to)
- 2) Issuing **Query** commands to find a given piece of content
- 3) Processing **QueryHit** responses and listing the nodes and content matching a given query
- 4) Actually downloading content via the **GET** operation using the HTTP protocol

You **aren't** responsible for implementing Gnutella topology management functions (Ping and Pong) nor are you responsible for implementing Push or any Gnutella protocol extensions. Finally, you are only responsible for downloading whole objects via HTTP GET (you don't need to implement range requests, pipelining, etc). While this assignment is probably less complex than project #1, it is also more open ended. There is no scaffold code for you to work from and you may need to be flexible in interpreting the specification to actually make your software work (real Gnutella clients sometimes

generate responses that are slightly different from the specification). This is effectively preparation for what working on networked software is like in the “real world”.

We aren't looking for major UI development in this assignment and you should just concentrate on basic functionality. To formalize this, you should simply construct two programs:

**gsearch** *ipaddress* *port* *n*

Should accept three arguments, **ipaddress** (in dotted decimal form), a **port** number and **n** (an integer). **ipaddress** and **port** are the IP address and TCP port number of the host in the Gnutella network you are connecting to. The argument **n** specifies the time your application should wait for responses to your queries before returning. Upon being executed, **gsearch** should accept a query (a string terminated by a newline) from standard input (stdin) and print formatted responses to standard output (stdout). The rough format of these responses should be:

**IP Address: port number\n**  
**File index/File name/File size\n**

Where there may be multiple File index/File name/File size lines under each IP address: port number pair. . For example:

```
ieng9.ucsd.edu% gsearch 132.239.17.78 1123 60
madonna

217.43.146.42:6346
423/Madonna - Hollywood.mp3/7060246
423/Prince & Madonna - Love Song
(Rare).mp3/4023146

68.71.18.215:6346
28/Madonna - Die Another Day (FULL).mp3/4275460
```

[In truth, you'd get far more responses than this in 60 seconds, this is just a hypothetical example]

**gget** *ipaddress* *port* *index* *filename*

Should accept four arguments, **ipaddress**, **port**, **index** and **filename**. **ipaddress** and **port** indicated the IP address and TCP port number respectively of the Gnutella node to be contacted. Similarly, **index** and **filename** indicate a node unique number and a filename to be downloaded. Together these four inputs will be combined to form an HTTP request for the object (as per the Gnutella

specification). The downloaded object should be written to a file named **filename** in the current working directory. If the object can't be found it should return an error message.

## Hints and Reminders

- 1) You may not use your software to violate UC policies (don't embarrass yourself or me please)
- 2) Both Gnutella and HTTP are based on the TCP protocol. You will be need to minimally use the socket calls `connect()`, `send()` and `recv()` to make this work. There are many great tutorials on socket programming on the Web (e.g., <http://www.ecst.csuchico.edu/~beej/guide/net/>).
- 3) If you don't want to "block" when receiving data you can use the `fcntl` call to mark a socket as non-blocking. Use "man `fcntl`" to get more information or use google. The call "`fcntl(sock, F_SETFL, O_NONBLOCK)`" will probably do the trick.
- 4) You're not responsible for manipulating the TTL field, but use a reasonable initial value (e.g. 7)
- 5) The "reference" Gnutella server we are using is `mutella` (which you are welcome to look at, but not take code from). We will be running a reference server at 132.239.17.78, port number 1123 that you can connect to. You can also find active gnutella servers at <http://www.gnucleus.net/gcache/gcache.php> (click on data).

Remember that you need to generate unique Gnutella Descriptor values for each query and match these values to subsequent responses (so you aren't mistakenly accepting responses to old queries). An easy way to do this is simply make the descriptor a combination of your IP address, userid, and the time of day. You're only responsible for managing one outstanding query at a time so don't get too complicated here.

## How to Proceed

Again, this assignment is somewhat more open ended than assignment one. We will not be providing a scaffold code base for you to work from. That said, here are some suggestions for how to proceed in completing this assignment.

- First, read the Gnutella protocol specification and review how HTTP works.
- Implement `gsearch` first
- First make sure you can connect to the Gnutella network (as per the protocol).
- Create a stub to print out each kind of Gnutella message you receive (this will be useful in debugging).
- Generate the query request and send it (you'll be able to tell if it works if you get `QueryHit` messages sent back to you)
- Parse and present the `QueryHit` results

- Make sure you wait for the proper amount of time (you can do this with non-blocking sockets, select or signals)
- Implement gget (this is a simple HTTP request). If you're unsure if you're doing the right thing you can emulate it by hand and see what happens (using telnet to connect to the appropriate host and port and typing the HTTP commands manually)

## Submission and Grading Guidelines

Make sure that all members of your team and their email addresses are contained at the top of each file. Put all of your files together in one directory (please delete any core files that may have been created). Make sure it compiles using a standard Makefile (you can adapt the file from project 1) or simply by compiling individual .c files (e.g. a single gsearch.c and gget.c) and test your solution. Create a single "tar" file containing all the files necessary to build your solution as follows: "tar cvf proj2.tar ." This will produce a single file called proj2.tar. Then electronically submit your assignment using turning:

```
turnin -c cs123b proj2.tar
```

We will compile and test your code in ieng9 so make sure it works in that environment (its not our job to debug portability problems)

The provisional grading is 10% for connecting to the gnutella network, 50% for sending queries and receiving and printing responses. 10% for correctly implementing and validating the descriptor id. 10% for implementing the timeout correctly (i.e. that you actually wait for n seconds worth of replies). 20% for implementing gget correctly. You're only responsible for interoperating with mutella (our reference server).