CS 640 Introduction to Computer Networks **Spring 2007**
http://www.cs.wisc.edu/~cs640-1

Assignment 4: Simple Name Server (SNS)

Assigned: April 17, 2007 *Due: April 24, 2007, 9:30am*

1 Introduction

The real name service in the Internet is used to resolve the mapping between a hostname and an IP address. In this assignment there are two entities that comprise the name service system — a name server and a resolver (client). You will implement a simplified version of *the resolver only*.

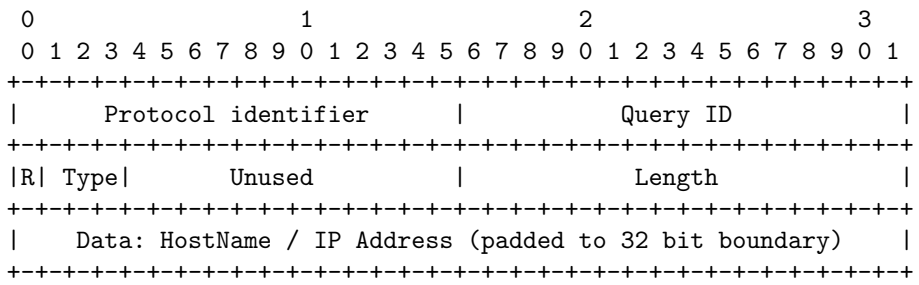
The server will run on a machine at a defined port number. The clients, called resolvers, will contact the server with queries about hostnames and IP addresses. On receiving a name service query, the server will lookup the query in its local database and respond with an appropriate answer. The server and the client (resolver) will use UDP — the User Datagram Protocol — to communicate with each other.

2 Protocol Specification

The protocol works as follows. The client will construct queries of different types and send them to the server. The server will query its internal database for each client query and send back a response. The client outputs this response and terminates. If a client does not get a response back from the server within a given timeout period, it assumes that the server is unreachable, outputs an appropriate error message, and terminates.

Packet Structure

Both the query and the response packets have the same header structure as shown below.



- **Protocol Identifier:** (2 bytes) Set to the integer value 640.
- **Query ID:** (2 byte) The value of this number is assigned by the client and is passed as an argument to the client program. The server must use the same ID. This way a client can distinguish between replies to his current query and duplicate/late replies to old queries.
- **R:** (1 bit) This is reserved for future use and can be either 0 or 1 in the messages. It is ignored by both the client and the server.
- **Type:** (3 bits) This indicates the type of the message.
  000 indicates IP address to hostname resolution request (sent by the client).
  001 indicates hostname to IP address resolution request (sent by the client).
  010 indicates IP address to hostname resolution response (sent by the server).

011 indicates hostname to IP address resolution response (sent by the server).
100 indicates that the server could not handle this request (sent by the server).
101 (sent by the client to the server) indicates that the server should terminate. Note that unless the server receives this message, the server should not terminate.

- **Unused:** (12 bits) These bits are ignored by the client and the server.
- **Length:** (2 bytes) The length of the packet (in bytes) including the header. by the client and the server.
- **Data:** (Variable length) This field contains the hostname or the IP address as would be required. For an IP address to hostname resolution request, this field contains the IP address; for a IP address to hostname resolution response, this field contains a hostname; etc.

  The IP address is stored in the network format as a 32-bit integer (see `inet_ntoa(3)` and `inet_aton(3)`. No padding is needed in this case. A hostname is stored using a NULL terminated text string, padded to a 32-bit word boundary.

The server will receive packets on a pre-defined port number. Both the server and the client will silently discard any packet which does not meet the specifications defined above. In particular, if the client receives a mal-formed packet it will discard this packet and continue to wait for a correctly formatted packet from the server, or until the timeout period is reached.

Upon receiving a valid query packet, the server will query its internal database for the information requested and will return what it finds in the database.

Note that the server will construct an appropriate response packet with its answer. For queries that do not have an entry in the database, the data part *will be zero bytes* and the packet type will be set to 100 (binary) as described above. For each query, the client uses a sequence number value which is passed to it as an argument. For a query with query ID x, the server will send its response with the same query ID. The client should verify that this is true in the response message that it receives.

If the server does not respond within some time period (specified by a timeout argument) the client will output an appropriate error message and terminate.

The server terminates on receiving a client request with the type field set to 101. The server does not send a response to the client in this case. The client that sends this request should also terminate immediately after sending this message.

3 Executables

The following is the expected executable format for the resolver (along with the necessary arguments):

- Resolver
  `client` [`-s` ⟨`server-ip`⟩] `-p` ⟨`port`⟩ `-r` ⟨`req-type`⟩ `-d` ⟨`data`⟩ [`-t`⟨ `timeout`⟩] [`-n` ⟨`query-id`⟩]

  server-ip: IP address where the server is running. Default is localhost.

  port: Port number at which the server is running.

  req-type: Is 0, 1, or 5. 0 is IP address to hostname resolution request. 1 is hostname to IP address resolution request. 5 is server termination request.

  data: Is the text argument associated with the request. For request type 5 (termination request), this field should be ignored.

  timeout: The time in seconds a client should wait before terminating after sending its request to the server. This field is ignored in request-type 5, since the client terminates immediately.

  query-id: The QueryID to be used for the request.

4 Output

On termination, the client should output the following:

- For IP address to hostname resolution request (req-type 0) output the hostname in a single line. For example:
  `machine.cs.wisc.edu`
- For hostname to IP address resolution request (req-type 1) output the IP address in a single line. For example:
  `128.56.49.12`
- In case the server does not provide any valid/correct response to the client prior to the timeout (note that all malformed responses are silently discarded), the following text in a single line:
  `timeout`
- In case the server responds to the query with the type field set to 100 (binary), i.e. the server could not resolve the request, the following text in a single line:
  `unresolved`
- For request type 5 (termination request), no output should be provided.

Please adhere exactly to this output format. For debugging purposes you may want to output other text messages. However, make sure you embed such messages with a DEBUG option and turn such output off in your final submission.

You should take care that your code is robust and handles possible erroneous messages from the server.

5 Submission

You will need to submit the source code along with a Makefile (located in a directory called `p1/`). in a single tar.gz file (name it p1.tar.gz). Do not submit object files, or compiled executables. The Makefile should have two rules: `clean` and `all`. `clean` will delete previous .o files, executables, etc. `all` should produce a single executable called `client`. The TA will run the following sequence of operations to execute and test your code:

```
tar xvfz p1.tar.gz
cd p1/
make clean
make all
./client -s <server-name> -p <port> -r <type> -d <data> -t <timeout> -n <query-id>
```

Please test to make sure that these commands can execute in sequence without any intervention. Further submission instructions will be posted in the class webpage.

6 Notes on Windows Development

If you decide to create your program under Windows, there are a few points you need to take care of

1. If you want to use the provided code outline, you will need to take the implementation of `getopt` from Gzip and add it to your project. You can find the required files on lots of websites, for example (`http://lottery.merseyworld.com/Wheel/`), you need to download `getopt.c`, `getopt.h` and `tailor.h`
2. At the start of your program, you need to add the following piece of code

```
WSADATA ws;
WSAStartup(0x0101,&ws);
```

to initialize the sockets library

3. Do not forget to add the library `wsock32.lib` in the "additional dependencies" section (under "Linker" in project settings)
4. The include files under Windows are a little different. You need to include "`winsock2.h`"