CS 640 Introduction to Computer Networks
Spring 2009
http://pages.cs.wisc.edu/~suman/courses/wiki/doku.php?id=640-spring2009

Programming Assignment 0: Simple Name Server (SNS)

Assigned: January 27, 2009 Due: February 10, 2009, 11:59pm

1 Introduction

The Internet's name service is used to resolve the mapping between a hostname and an IP address. There are two sets of entities that comprise the name service system – name servers and resolvers (clients). You will implement a simplified version of the *solver only*.

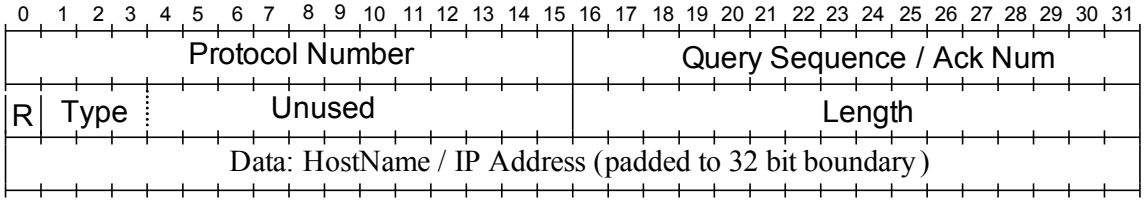
The server will run on a machine at a defined port number. The clients, called resolvers, will contact the server with queries about hostnames and IP addresses. On receiving a name service query, the server will lookup the query in its local database and respond with an appropriate answer. The server 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. Depending upon the response, the client outputs the 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 Sequence/Ack Number**: (2 bytes) The value of this sequence number is assigned by the client and is passed as an argument to the client program. On its response, the server increments it by one.

- **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).
  110 indicates that the server should terminate. Note that unless the server receives this message, the server should not terminate. (sent by the client to the server). Note that it is only in this assignment that the clients can terminate the server. In the real Internet's name service the clients do not have this privilege.

- **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.

- **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 sent by the client and the server.

- **Data**: (Variable length) Depending upon the packet type, this field can have an IP address (000, 011), hostname (001, 010) or IP address and port number (101). 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; for a redirect response it contains the IP address and port number of the server that should be contacted next by the resolver. The IP address is stored in the network format as a 32-bit integer (see inet ntoa() and inet aton()). The port is stored in the network format as a 16-bit integer (see htons() and ntohs()). No padding is needed in this case. A hostname is stored using a NULL terminated text string, padded to a 32-bit word boundary.

- **Note**: The numbers such as protocol, sequence number, length, and port in the packet data must be network order. It means that you must use htons() function to do the transformation before you pack the number into the packet.

- **Procedure**:

  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 sequence number x, the server will send its response with an acknowledgment number value set to x+1. 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.

4 Executables

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

- **Resolver**

client [-s<server-ip>] -p <port> -r <req-type> -d <data> [-t <timeout>] [-n <seq-no>]

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 6. 0 is IP address to hostname resolution request. 1 is hostname to IP address resolution request. 6 is server termination request.
data: Is the text argument associated with the request. For request type 6 (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 6, since the client terminates immediately.
seq-no: The sequence number to be used for the request.

5 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 6 (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.

6 Submission

You will need to submit the source code along with a Makefile (located in a directory called p0/). in a single tar.gz file (name it p0.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 p0.tar.gz
cd p0/
make clean
make all

./client -s <server-name> -p <port> -r <type> -d <data> -t <timeout> -s <seq-no>

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.