CS 324, Fall 2017
DNS Resolver Lab
Due: Friday, Nov. 3, 11:59PM

1 Overview

This lab will provide you a basic understanding of socket programming and will help you understand the complexities and importance of Internet protocols and agreement.

In this lab, you will be building a basic *DNS stub resolver*. A stub resolver is the library on your computer that an application, such as your Web browser or email client, uses to translate a domain name, e.g., `www.example.com` to an IP address, e.g., `192.0.2.1`.

The DNS resolver is considered a *stub* because the extent of its functionality is to:

- format the question into a DNS query of the proper format;

- send the query to a designated *recursive* DNS server;

- wait for and receive the response from the server; and

- extract the answer from the response and return it to the caller.

The recursive DNS server does the hard work of tracking down the answer by issuing its own queries to (sometimes many) other DNS servers.

In this lab the effort will be in communicating effectively with an actual DNS server by implementing the protocol.

2 Downloading the assignment

The files for the lab will made available from a link in the "assignment" page for the lab on the course site on Learning Suite, as a single archive file, `dnsresolverlab-handout.tar`.

Start by copying `dnsresolverlab-handout.tar` to a protected Linux directory in which you plan to do your work. Then give the command

```
linux> tar xvf dnsresolverlab-handout.tar
```

This will create a directory called `dnsresolverlab-handout` that contains a number of files. You will be modifying the file `resolver.c`.

You can build the executable program by running the following:

```
linux> make clean
linux> make
```

The program is run by supplying the domain name and server as command-line arguments, like this:

```
linux> ./resolver www.example.com 8.8.8.8
www.example.com => 93.184.216.34
```

(Note that `8.8.8.8` is a public DNS recursive resolver. Another server to test against is the BYU CS DNS resolver, `128.187.80.20`.)

Its output should match that of the reference implementation, `resolver-ref`:

```
linux> ./resolver-ref www.example.com 8.8.8.8
www.example.com => 93.184.216.34
```

3 Description

There are various tasks associated with building a DNS stub resolver.

3.1 Building a DNS Query Message

The first task is organizing the various components of the DNS query into a *DNS query message*. The query message is simply an array of `unsigned char`, which can be transmitted to the server over a UDP socket. Building the contents of that array is a matter of organizing and formatting the query components according to the DNS protocol specification. There are two major parts to the DNS query message: the DNS header and the DNS question section, which contains the query.

The DNS header and query are organized as shown in the following page:

`http://www.networksorcery.com/enp/protocol/dns.htm`

in the sections under "DNS header:" and "Query. Variable length", respectively.

For example, a query for `www.example.com` looks like this:

```
27 d6 01 00
00 01 00 00
00 00 00 00
03 77 77 77
```

```
07 65 78 61
6d 70 6c 65
03 63 6f 6d
00 00 01 00
01
```

This is equivalent to:

```
unsigned char msg[] = {
0x27, 0xd6, 0x01, 0x00,
0x00, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x03, 0x77, 0x77, 0x77,
0x07, 0x65, 0x78, 0x61,
0x6d, 0x70, 0x6c, 0x65,
0x03, 0x63, 0x6f, 0x6d,
0x00, 0x00, 0x01, 0x00,
0x01
};
```

In the above example, the following are the header values:

- Identification (query ID): `0x27d6` This value is arbitrary and is typically randomly-generated when the query is formed (note that network protocols use *network byte order*, which is equivalent to Big-endian (most significant bytes are on the left).

- QR flag: `0` (query)

- Opcode: `0` (standard query)

- Flags: The `RD` bit is set (`1`); all others are cleared (`0`).

- Total questions: `1` (again, note the network byte ordering)

- Total answer RRs: `0`

- Total authority RRs: `0`

- Total additional RRs: `0`

And the question section contains the query:

- Query name: `www.example.com`, encoded as follows (encoding method shown later):

```
03 77 77 77
07 65 78 61
6d 70 6c 65
03 63 6f 6d
00
```

- Type: `1` (type `A` for address)

- Class: `1` (`IN` for Internet class)

Note that in this lab, you will *only* be issuing queryies for type `1` (`A`) and class `1` (`IN`). The header will largely look the same for all queries produced by your resolver—with only the Query ID changing (to random values). Similarly, the Question section will look the same, withonly the query name changing.

The `create_dns_query()` function has been declared in `resolver.c` and is intended as a helper function to build the `unsigned char` array which will be transmitted to the server. Other helper functions include `name_ascii_to_wire()` and `rr_to_wire()`, also declared in `resolver.c` but left as an exercise to you to define. Note also the `typedef` declarations for `struct dns_rr` and the various DNS fields (e.g., `dns_rr_type`).

The following functions might be useful for your message composition: `strok()`, `strlen()`, `strncpy()`, `memcpy()`.

3.2 Sending a DNS message and Receiving a Response

A well-formatted DNS query in an array of `unsigned char` can be transmitted to a DNS server. The `send_recv_message()` helper function is declared to help with sending the message and receiving the response. Note that the protocol will be UDP (type `SOCK_DGRAM`), and the destination port is `53` (the standard, well-known DNS port).

Useful structures include the `struct sockaddr_in` data structure (see `man inet_pton`), `socket()` (to create a socket), `connect()` (to "connect" to a remote server, prior to sending/receiving data to/from that server), `send()` (to send data to a socket), `recv()` (to receive data from a socket), `htons()` (to convert a short to network order), `inet_addr(server)` (for converting a string IP address to bytes). See the `man` pages for each of these functions.

3.3 Extract an Answer from a DNS Response

Having received a DNS message response from the server, stored in an array of `unsigned char`, the next step is to extract and decode the useful information (i.e., define the `get_answer_address()` function). For the purposes of this lab, the useful parts are the answer count and the resource records in the answer section. You will want to extract each resource record from the section and determine whether or not it is the one you are looking for. Specifically, you will implement the following algorithm (provided in pseudo code):

```
set qname to the initial name queried
    (i.e., the query name in the question section)
  for each resource record (RR) in the answer section:
      if the owner name of RR matches qname and the type matches the qtype:
          extract the address from the RR, convert it to a string, and return it
      else if the owner name of RR matches qname and the type is (5) CNAME:
          the name is an alias; extract the canonical name from the RR rdata,
```

```
        and set qname to that value
return NULL (no match was found)
```

The resource records in the answer section are different than the Query in the Question Section, in that they include a time-to-live (TTL) value, Rdata (e.g., the IP address for resource records of type A), and Rdata length:

`http://www.networksorcery.com/enp/protocol/dns.htm`

See the section under "Resource Record. Variable length".

A response to a query for `www.example.com` might look like this:

```
27 d6 81 80
00 01 00 01
00 00 00 00
03 77 77 77
07 65 78 61
6d 70 6c 65
03 63 6f 6d
00 00 01 00
01 c0 0c 00
01 00 01 00
01 01 82 00
04 5d b8 d8
22
```

This is equivalent to:

```
unsigned char msg[] = {
0x27, 0xd6, 0x81, 0x80,
0x00, 0x01, 0x00, 0x01,
0x00, 0x00, 0x00, 0x00,
0x03, 0x77, 0x77, 0x77,
0x07, 0x65, 0x78, 0x61,
0x6d, 0x70, 0x6c, 0x65,
0x03, 0x63, 0x6f, 0x6d,
0x00, 0x00, 0x01, 0x00,
0x01, 0xc0, 0x0c, 0x00,
0x01, 0x00, 0x01, 0x00,
0x01, 0x01, 0x82, 0x00,
0x04, 0x5d, 0xb8, 0xd8,
0x22
};
```

In the above example, the following are the header values:

- Identification (query ID): `0x27d6` Matches the query ID of the query.

- QR flag: `1` (response)

- Opcode: `0` (standard query)

- Flags: The `RD` and `RA` bits are set (`1`); all others are cleared (`0`).

- Total questions: `1`

- Total answer RRs: `1`

- Total authority RRs: `0`

- Total additional RRs: `0`

The question section is identical to that of the DNS query. The answer section contains a single resource record with the following values:

- Owner name: `www.example.com`, encoded using compression encoding (encoding method shown later):

  ```
  c0 0c
  ```

- Type: `1` (type `A` for address)

- Class: `1` (`IN` for Internet class)

- Rdata length: `4` (and IP address is four bytes)

- Rdata: The bytes comprising the IP address corresponding to the owner name

The `name_ascii_from_wire()` and `rr_from_wire()` helper functions can be defined by you to help with this process. Additionally, the externally-defined functions `malloc()`, `memcpy()`, `inet_ntop()`, and `strcmp()` might be useful.

3.4 Name Encoding and Decoding

Encoding a domain name for a DNS message is explained here:

`http://www.tcpipguide.com/free/t_DNSNameNotationandMessageCompressionTechnique.htm`

The DNS query has an example of this, encoding `www.example.com` thus:

```
03 77 77 77 07 65 78 61 6d 70 6c 65 03 63 6f 6d 00
   w  w  w     e  x  a  m  p  l  e     c  o  m
```

The length of each label precedes the characters comprising the label itself (e.g., the value `3` precedes the three bytes representing the ASCII values `www`). A `0` in the length byte always indicates the end of the labels.

Decoding is similar, but there is a special case called *compression*, which is explained here:

```
http://www.tcpipguide.com/free/t_DNSNameNotationandMessageCompressionTechnique-2.htm
http://www.tcpipguide.com/free/t_DNSNameNotationandMessageCompressionTechnique-3.htm
```

Note that the DNS response above uses compression for `www.example.com`:

```
c0 0c
```

When the two most significant bits (bits `6` and `7`) are set in the first byte—and thus the value is ≥ 192—it is an indicator that the byte doesn't represent length; rather it indicates that the *next* byte is a pointer (the offset in the DNS message) to where the sequence of next labels for the name are found, terminated by a "length" byte with value `0`. In the above case, the byte values are `0xc0` and `0x0c`. Thus, the first indicates that the second is a pointer ($0xc0 = 192_d \geq 192_d$), and the byte value of the second, `0x0c` (`12` in decimal), is the offset at start of the question section. Indeed the owner name of the answer resource record is the same as the query name!

3.5   Other Notes

Note that you don't have to use the declared helper functions. The only required function to be defined as declared is `char *resolve()`. But we hope that the declarations and descriptions in the source will help you on your way.

There are two functions, `print_bytes()` and `canonicalize_name()` that are provided for your benefit to help with development and troubleshooting.

The encoding of the *root* name, represented in string form as ".", is a single byte, `0x00` (note that this is the final byte in encodings of all other domain names because the root is the ancestor of all domain names).

Sometimes there are multiple IP addresses returned for a given name (see `byu.edu`, for example). For the purposes of this lab, only the first IP address needs to be returned. The others can simply be ignored.

You might find it useful to use the command-line tool, `dig` to issue queries against servers and see a more textual representation of the responses. For example:

```
linux> dig @8.8.8.8 www.example.com
```

4   Testing

You are invited to test your code against multiple DNS resolvers, including `8.8.8.8` and `128.187.80.20` (from the CS network). The following domain names can be used to test:

- `byu.edu` (simple response)

- `www.byu.edu` (single `CNAME` record, single domain)

- `i-dont-exist.byu.edu` (name that doesn't exist—no answer records)

- `www.intel.com` (multiple `CNAME` records in response, multiple domains chased for compression)

- `.` the root domain name (note that it won't have any `A` records, but it shouldn't bomb out)

5   Grading

The following is the point breakdown:

- - 10 points for a well-formed DNS query message

- - 10 points for successfully sending the query and receiving the response

- - 20 points for successfully finding the answer in the answer section

- - 5 points for handling CNAME records properly

- - 5 points for handling names that don't resolve

- - 5 points for handling the root name properly

- - 5 points for style

The maxmimum number of points is 60.

6   Handing in Your Work

To submit your lab, please upload your `resolver.c` file to the assignment page corresponding to the DNS resolver lab on Learning Suite. Also, upload your files to the CS open lab machines, compile and test your code there, then run 'make handin' on the CS open lab machines to copy your code to the handin directory.