Patt-Shamir

Programming Assignment 1: DNS Client

Due: April 22, 2020. In this assignment you will implement a simple Internet protocol: you will write a UDP client working against DNS servers. You will experiment the unique style of RFCs, and get a first-hand experience of writing, reading and interpreting message bits.

1 User Interface

In this assignment you will implement a simple DNS client: a streamlined version of the *nslookup* utility. (Try it out! nslookup is implemented as a command line utility in Windows; in the Unix family, use *host* or the very informative *dig* utility.)

The program gets a single command line argument which is the IP address of a DNS server, given in dotted-decimal notation. Thereafter, it prompts the user to enter a string of non-blank characters. The string is verified syntactically, and if looks fine, it is sent to the server as a domain name (otherwise the client locally reports an error). The result is printed on the screen in dotted-decimal notation. The process exits when a line containing the single word "quit" is reached. A typical usage of the utility may look as follows (note the different reaction of the program to a malformed query and to a non-existent domain).

```
C:\networks\assignment1> nsclient 132.66.48.2          shell command line with server IP address
nsclient> bakara.eng.tau.ac.il                         program prompts, user inputs
132.66.48.12                                           program's responses, and then prompts
nsclient> zooot.tau.ac.il
ERROR: NONEXISTENT                                     another possible response
nsclient> zoot.tau.ac.il
132.66.16.59
nsclient> *%213kj4h&oh0970987y12
ERROR: BAD NAME
nsclient> quit                                         exit program
C:\networks\assignment1>                               program finished: shell prompt
```

2 Program structure

Your assignment is to hand in the complete program that does not use *gethostbyname* or its variants. Your main program will call a function named *dnsQuery* that implements a subset of the functionality of *gethostbyname*, and has the same interface: it accepts a NULL-terminated string as an argument,

and returns a pointer to a static struct hostent. The main program does the user interface, and the *dnsQuery* function talks to the DNS server.

The dnsQuery function forms a dns query message according to the RFC specification, and sends it in a UDP message (using *sendto*) to port 53 of the server machine (specified in the command line). It waits for an answer in *recvfrom*, but no more than 2 seconds. When an answer arrives, it is parsed, the hostent structure is filled, and a pointer to it is returned. If no answer is received within 2 seconds, the user is notified and a null pointer is returned.

The actual DNS queries you are requested to do are just the basic domain-name to IP-address translation, as described in RFC 1035. Some excerpts from the RFC appear in the appendix; the full document can be found easily on the Internet (e.g., http://faqs.org/rfcs/rfc1035.html).

- You must check the result of each system call. Use *perror*!
- You may wish to write and debug the main program first, using the standard *gethostbyname*.
- Domain names are *not* null terminated strings. In the query (and in the answer resource records), they are encoded as a sequence of the name components ending with a 0-length component. Each component is encoded by a leading byte indicating the length of the component, followed by a sequence of characters. For example, the domain name `zoot.tau.ac.il` is encoded by the following sequence of bytes:

  ```
  4, 'z', 'o', 'o', 't', 3, 't', 'a', 'u', 2, 'a', 'c', 2, 'i', 'l', 0
  ```

  Upper and lower case letters are equivalent: ZoOt.tAU.Ac.IL=zoot.tau.ac.il.
- DNS queries: The ID is used to distinguish between multiple queries sent from the same socket. Typically, all header fields after the ID and before the QDCOUNT set to 0. The ID can be generated sequentially. You may ignore the authority issue.
- DNS answers: Obviously, the QR field is 1. The RCODE field must be checked.
- Using wireshark may save you a lot of time in debugging! See http://www.wireshark.org/.

3 How to Test

We advise you to test your implementation using your regular DNS server. To find which is it, you can type `ipconfig /all` in a windows command shell, and look for the "DNS Servers" section in the output, or, in Unix shells (including Linux and MacOS), look for the "nameservers" line(s) in the file */etc/resolv.conf*, say by typing `grep nameserver /etc/resolv.conf`.

4 What to submit

Submission will be done by via Moodle. Instructions were posted in the course Moodle site. Since this is the first assignment, we repeat the instructions here.

You need to submit a ZIP archive that contains the following files, *and them only*:

- README file: contains the full documentation, including names of the team members. We will read this file. The documentation should describe whatever is unusual about your implementation (such as additional or missing features). Include brief explanation of your algorithm and data structures in the code.
- Visual studio project (as explained in the course site): used to build your application under the name `nsclient`.
- All project .h and .c (or .cpp) source code files.

The grader will unzip the archive, build your program using the .dsw file, and then type "`nsclient` *`server-name`*". Make sure that your submission will work when this simple procedure is applied!

The grader will test the program against a few typical names, and will also try the program with syntactically malformed names, and with syntactically correct but non-existent names. In addition, we will read your documentation and look at the source code. Keep a good coding style!

5 References

1. *Unix Network Programming*, W. Richard Stevens. The definitive text for its topic. *Beej's Guide* is a readable tutorial about socket programming (see "resources" page in the course site).
2. MSDN pages for nslookup, gethostbyname, socket, bind, sendto, recvfrom. All these pages are available on-line from http://msdn.microsoft.com.

APPENDIX: Excerpts from RFC 1035

All communications inside of the domain protocol are carried in a single format called a message. The top level format of message is divided into 5 sections (some of which are empty in certain cases) shown below:

```
+---------------------+
|        Header       |
+---------------------+
|       Question      | the question for the name server
+---------------------+
|        Answer       | RRs answering the question
+---------------------+
|      Authority      | RRs pointing toward an authority
+---------------------+
|      Additional     | RRs holding additional information
+---------------------+
```

The header section is always present. The header includes fields that specify which of the remaining sections are present, and also specify whether the message is a query or a response, a standard query or some other opcode, etc.

The names of the sections after the header are derived from their use in standard queries. The question section contains fields that describe a question to a name server. These fields are a query type (QTYPE), a query class (QCLASS), and a query domain name (QNAME). The last three sections have the same format: a possibly empty list of concatenated resource records (RRs). The answer section contains RRs that answer the question; the authority section contains RRs that point toward an authoritative name server; the additional records section contains RRs which relate to the query, but are not strictly answers for the question.

5.1 Header section format

The header contains the following fields:

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

ID A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.

QR A one bit field that specifies whether this message is a query (0), or a response (1).

OPCODE A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are:

- 0 a standard query (QUERY)
- 1 an inverse query (IQUERY)
- ...

AA Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.

TC TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.

RD Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.

RA Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server.

Z Reserved for future use. Must be zero in all queries and responses.

RCODE Response code - this 4 bit field is set as part of responses. The values have the following interpretation:

- 0 No error condition
- 1 Format error - The name server was unable to interpret the query.
- 2 Server failure - The name server was unable to process this query due to a problem with the name server.
- 3 Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.
- 4 Not Implemented - The name server does not support the requested kind of query.
- 5 Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data.
- 6-15 Reserved for future use.

QDCOUNT an unsigned 16 bit integer specifying the number of entries in the question section.

ANCOUNT an unsigned 16 bit integer specifying the number of resource records in the answer section.

NSCOUNT an unsigned 16 bit integer specifying the number of name server resource records in the authority records section.

ARCOUNT an unsigned 16 bit integer specifying the number of resource records in the additional records section.

5.2 Question section format

The question section is used to carry the "question" in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                     QNAME                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     QTYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     QCLASS                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

QNAME a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.

QTYPE a two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.

QCLASS a two octet code that specifies the class of the query. For example, the QCLASS field is IN for the Internet.

5.3 Resource record format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                                               /
/                      NAME                     /
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

NAME a domain name to which this resource record pertains.

TYPE two octets containing one of the RR type codes. This field specifies the meaning of the data in the RDATA field.

CLASS two octets which specify the class of the data in the RDATA field.

TTL a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached.

RDLENGTH an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record. For example, the if the TYPE is A and the CLASS is IN, the RDATA field is a 4 octet ARPA Internet address.

5.4 Constants

TYPE	value	meaning
A	1	a host address
NS	2	an authoritative name server
...		
TXT	16	text strings
AXFR	252	A request for a transfer of an entire zone

QTYPEs		
...		
*	255	A request for all records

CLASS		
IN	1	the Internet
...		
*	255	any class

5.5 Message compression

In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurance of the same name.

The pointer takes the form of a two octet sequence:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 1  1|                OFFSET                   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The first two bits are ones. This allows a pointer to be distinguished from a label, since the label must begin with two zero bits because labels are restricted to 63 octets or less. (The 10 and 01 combinations are reserved for future use.) The OFFSET field specifies an offset from the start of the message (i.e., the first octet of the ID field in the domain header). A zero offset specifies the first byte of the ID field, etc.

The compression scheme allows a domain name in a message to be represented as either:
- a sequence of labels ending in a zero octet
- a pointer
- a sequence of labels ending with a pointer

Pointers can only be used for occurances of a domain name where the format is not class specific. If this were not the case, a name server or resolver would be required to know the format of all RRs it handled. As yet, there are no such cases, but they may occur in future RDATA formats.

If a domain name is contained in a part of the message subject to a length field (such as the RDATA section of an RR), and compression is used, the length of the compressed name is used in the length calculation, rather than the length of the expanded name.

Programs are free to avoid using pointers in messages they generate, although this will reduce datagram capacity, and may cause truncation. However all programs are required to understand arriving messages that contain pointers.

For example, a datagram might need to use the domain names F.ISI.ARPA, FOO.F.ISI.ARPA, ARPA, and the root. Ignoring the other fields of the message, these domain names might be represented as in figure 1. The domain name for F.ISI.ARPA is shown at offset 20. The domain name FOO.F.ISI.ARPA is shown at offset 40; this definition uses a pointer to concatenate a label for FOO to the previously defined F.ISI.ARPA. The domain name ARPA is defined at offset 64 using a pointer to the ARPA component of the name F.ISI.ARPA at 20; note that this pointer relies on ARPA being the last label in the string at 20. The root domain name is defined by a single octet of zeros at 92; the root domain name has no labels.

```
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
20 |           1           |           F           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
22 |           3           |           I           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
24 |           S           |           I           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
26 |           4           |           A           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
28 |           R           |           P           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
30 |           A           |           O           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
40 |           3           |           F           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
42 |           O           |           O           |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
44 | 1  1|                 20                      |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
64 | 1  1|                 26                      |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
92 |           0           |                       |
   +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Figure 1: Example.