VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Network applications and management - project documentation

variant: DNS resolver

19.11.2023

Vojtěch Kališ (xkalis03)

Obsah

1 Introduction

The project's task was to create a C-based or C++-based implementation aiming to provide a functional tool for Domain Name System (DNS) resolution; this specific implementation is written in C. Its focus was on creating a functional resolver capable of DNS queries creation in the form of UDP packets, establishing communication with given DNS server, successful sending of said packet, and subsequent retrieval and processing of the response information.

1.1 Program arguments

usage: dns [-r] [-x] [-6] -s server [-p port] address

[-r] = recursion desired

[-x] = make reverse request instead of direct request

[-6] = make request of type AAAA instead of default A

-s server = IP or hostname of server to which request will be sent

[-p port] = port number to use

address = address that is the object of query(request)

The arguments hereby listed were all provided by the project specification. However, some specific interactions weren't clearly implied and hence were up to interpretation—such as follows:

- Reverse request (**-x**) requires *server* to be an IP address, not hostname. While some workaround could mayhaps be found, it was instead chosen to throw an error should this scenario arise.
- Reverse request (**-x**) and request of type AAAA (**-6**) are incompatible, as the former requires query type to be set to PTR, while the latter wishes to set it to AAAA. Once again, it was chosen to throw an error in this instance as well.

1.2 Software requirements

- Unix-like operating system, such as Linux or macOS
- A C compiler supporting the C standard libraries and functionalities
- Network connectivity
- Python version 3 for running tests

2 Resources & Application layout

Starting up (and further implementation of) the project involved a lot of web browsing and research. Eventually, this basic implementation [3] providing the basic functionality for DNS resolving was stumbled upon, and had been stripped and used as the bare bones of the project. The RFC 1035[2] was then used to restructure the DNS packet header, query and response structures, though in case of header, it has been opted to use structure as shown here[1] instead. RFC 3596[5] then served well to implement IPv6 functionality as well, especially so when reverse DNS request was concerned. DNSname and hostname conversions were based on functions which can be found in this implementation [4]

2.1 Application layout (dns.c)

It is important to note that, while the project consists of a multitude of functions all serving their specific purposes, the subject of this section is to provide overview of the main functions; that is, functions which have the biggest direct contribution to the subject of the project—DNS resolving itself. These functions can be found in file *dns.c* underneath the *INTERNAL PROGRAM FUNCTIONS* header.

In case of interest in the other, auxiliary functions, the header file *dns.h* provides a basic review of every single function contained within the program, and an effort has also been made to provide as much commentary inside the main file *dns.c* as possible.

2.1.1 parse_args

Parses command-line arguments to determine query parameters. It is here where the vast majority of invalid inputs are detected and handled (program exit, corresponding errors thrown)

2.1.2 sock_prep

Sets up sockets for communication with DNS servers, including setting timeouts and preparing IPv4 or IPv6 addresses.

2.1.3 dns_pack_prep

Constructs DNS packet headers with default values, enabling modifications for specific query types and flags.

2.1.4 dns_qname_insert

Handles the transformation of domain name into DNS name format part of DNS query preparation.

2.1.5 dns_qinfo_prep

Handles the query information configuration (setting Qtype and Qclass) part of DNS query preparation.

2.1.6 dns_reply_load

Takes DNS reponse packet and processes it by populating pre-prepared arrays for answer, authority and additional records with respective data

2.1.7 main

Orchestrates the entire DNS resolution process, utilizing the above functions in their respective order to handle parsing input arguments, creating a DNS packet, setting up socket and then sending prepared query and receiving the answer. Lastly, it processes the response and then prints it out in required format using the *project_print* function.

3 Functionality

The program starts its function inside the *main* function, where it first calls function *parse_args* to parse input arguments. Then, it creates a packet message buffer large enough to contain the entire response, after which comes the task of preparing a socket to connect to the desired DNS server; this icludes creating the socket and socket address structures, sending them to the *sock_prep* function to set said packet up for UDP packet (for DNS queries) and prepare the IPv4 or IPv6 address for the query.

After the socket is prepared, the program creates a DNS packet header structure and points it to its appropriate location within the buffer, then utilizes the *dns_pack_prep* function to initialize all header bits and sets the *recursion desired* bit based on if it's desired or not (meaning if '-r' argument was received or not).

Now comes the part of preparing the dns query structure—resolving query hostname, converting it into DNSname for which the *dns_qname_insert* function is used, and inserting it into the buffer right after the DNS header structure. After that, the program also creates a query structure (holding the query type and class), inserts it into buffer after the query name, and fills it up.

With the packet structure now prepared, the program attempts to send the packet to the desired server and then attempts to retrieve the response (by loading it into the buffer, letting it overwrite our configuration). A pointer to the portion of the packet after all previously mentioned headers pushed into the buffer gets set and sent to the *dns_reply_load* function, which then starts reading the reply from then on and loading each individual record into pre-prepared structure of record string arrays.

Last but not least, the program passes all necessary data pointers and the records structure into a special *project_print* function, which prints information out to stdout as per the task's requirements, and then utilizes the *clean_exit*function to free any and all allocated memory.

Used literature

[1] CyberCash, C. Kaufman, and Iris. The ad and cd header bits. https://www.freesoft.org/CIE/RFC/2065/40.htm, 1997.

[2] P. Mockapetris. Domain names - implementation and specification. https://datatracker.ietf.org/doc/html/rfc1035, 1987.

[3] Silver Moon. Dns query code in c with linux sockets. https://www.binarytides.com/dns-query-code-in-c-with-linux-sockets/, 2020.

[4] riveraj. Dns resolver. https://github.com/riveraj/dns-resolver/blob/master/main.c, 2013.

[5] S. Thomson, Cisco, C. Huitema, Microsoft, V. Ksinant, 6WIND, M. Souissi, and AFNIC. Dns extensions to support ip version 6. https://datatracker.ietf.org/doc/html/rfc3596, 2003.