CSE30: Computer Organization and Systems FA20
Assignment 4
Instructors: Leo Porter, Keith Muller

Background (Optional Reading)

In the area of network security and privacy there is a category of devices that focus on network tracking and advertisement blocking called DNS sinkholes.

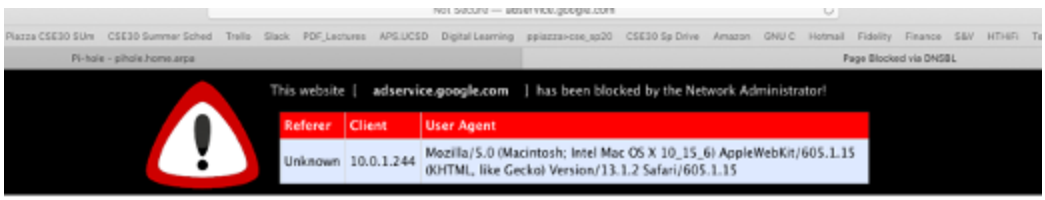
From Wikipedia (https://en.wikipedia.org/wiki/Domain_Name_System): “The **Domain Name System** (**DNS**) is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. The DNS system associates various information with domain names assigned to each of the participating entities. One important very important function of DNS is that it translates more readily memorized domain names, such as www.ucsd.edu, to the numerical IP addresses needed for locating and identifying computer services and devices with the underlying network protocols. By providing a worldwide, distributed directory service, the Domain Name System has been an essential component of the functionality of the Internet since 1985.”

When your computer connects to the network, it is given the IP address of a DNS server where DNS queries are to be sent. An end-user application, like a web browser, will make a DNS request for every DNS name associated with a web page that you visit. The DNS server will respond with the IP address for each DNS name.

The open source project, pihole (https://pi-hole.net/), turns a raspberry pi into a DNS server (and DNS sinkhole) for your local/home network. Computers on your local network are then configured to make DNS requests directly to the pihole DNS server. The pihole DNS server will either respond to your DNS name request directly (it keeps copies of commonly requested domain names) or will query higher level DNS servers on your behalf to get the IP address and then forward the response to your computer. By placing the pihole DNS server between your computer and the internet DNS servers allow the pihole to filter DNS requests to block sites that you do not want to visit (like advertising, tracking sites, malware etc).

A DNS sinkhole includes a database of sites keyed by DNS names that are to be blocked. When a DNS sinkhole receives a DNS request for a blocked site from say a web browser on your computer, it responds not with the actual IP address of the requested site but substitutes the IP address of the DNS sinkhole. The application attempts then attempts to connect to the DNS sinkhole instead. The DNS sinkhole takes the place of the real site and in the case of a web page connection it returns a blank page (blocking access to the actual page requested), or for other types of applications drops the request.

In the picture below we show what this looks like at a client web browser when the IP address for a blocked address (adservice.google.com) is returned. The warning page from the DNS sinkhole web server replaces the actual web page.



Assignment Overview

Your programming assignment is to implement a hash table-based in-memory database using single linked chains for collision resolution. The database will contain DNSnames of websites you wish to block. You will load a real DNS block list into the database and then make some queries for different DNS names. If a DNS name is found in the database it will respond with a blocked message. If the DNS name is not in the database it will respond with a not blocked message.

Part 0: Getting started

We've provided you some starter code with a Makefile here. Download all the files to use.

1. Download to get the Makefile, dnsblock.c, blocklist.txt, and test_file.txt
2. Open your favorite text editor or IDE of choice and start the assignment

*The files you turn in must be named as specified or else the autograder and makefile provided will not work.

Part 1: dnsblock program [9 points]

Step 1. Command line Parsing

Your program will be called using the following syntax:

```
dnsblock [-s] [-t tablesize] -b blockfile
```

The **-s** option prints out the following information after the hash table has been loaded. It works by walking down each chain to find the total number of entries and the length of the longest and shortest collision chain.

```
Table size: 1873
Total entries: 75064
Longest chain: 65
Shortest chain: 18
```

The **`-t tablesize`** option allows an override of the default size of the hash table array (use 1873 as a default) to `tablesize`. Making this a small positive number may help you to debug your chaining routines. The simple hash function we are using (see below) often works better with prime numbers for the size of the hash table.

The **`-b blockfile`** must be specified as it is the filename of the block list test file to load into the database. If not specified it is an error. You will open the file using `fopen()` tin read only mode ("r") read the file with `getline()` and the `fclose()` the file after EOF is reached.

Step 2. Load the Hashtable from the blockfile

Each entry in the hash chain is of the following type:

```
typedef struct node {
        char *DNSname;
        struct node *next;

    } node;
```

Create the hash table on the head where htable is a pointer to the table;

node **htable;

Using `calloc()`, allocate the space for the hash table to contain `tablesize` elements of (node *). This way the table is zero filled (where all the (node *) pointers are NULL to start;.

The blockfile is an ascii text file of blocked DNS names. Each line of the file will contain a single DNS name terminated by a newline ('\n'). Lines that start with a # or a '\n' are skipped over (ignored - not inserted into the hash table). You can assume all other lines are ok. Until you reach the end of the file, read the text file, one line at a time using a `getline()` loop, remove the trailing '\n' from the input with (you can use a routine like `strchr()if you like)` and then generate a hash value for each entry in the block DNS file.

For your hash function (from Dan Bernstein, University of Illinois) use the following code:

```
unsigned long hash(char *str)
{
    unsigned long hash = 0;
    unsigned int c;

    while ((c = (unsigned char)*str++) != '\0')
        hash = c + (hash << 6) + (hash << 16) - hash;

    return hash;
}
```

Make sure you use unsigned long for the hash value as shown. Using the hash value, you will check if the entry is already in the table. If the DNSname you are trying to add to the database is already in the table, using `fprintf()` you will print a message line (as below) to stderr and go to the next entry in the blockfile.

Loadtable Duplicate entry: ads.google.com

If this DNSname is not in the file, you need to insert it into the hash table. Use `malloc()` to allocate a **node** , and use `strdup()` to allocate the space (pointed at by a DNSname member) to contain a DNSname string. Insert the new entry at the front of the collision chain at the bucket specified by the hash value.

Step 3. Print the stats

If the `-s` option is given you will walk the hash table from the first chain pointer (offset 0) to the last (offset `tablesize` – 1). For each entry you walk the chain and while doing so count the number of entries on the chain. You need to keep track of the shortest and longest chain as well as the total number of entries in the table.

Step 4. Query Mode

After the table is built, you will read standard input (until a ctrl-d is pressed on a line by itself and the program will exit) using `getline()`. You can type on each line input a DNS name that you will check to see if it is blocked (found in the hash table) or is not blocked (not found in the hash table). Lines that start with a # or a '\n' are skipped over. You can also create a test file and test it with:

```
dnsblock -s -b blocklist.txt < test_file.txt
```

Your output will be DNS name input followed by a space and either [blocked] or [not blocked]:

Below we provided the output of executing the program with test_file.txt file that was provided as part of the starter code.

```
Table size: 1873
Total entries: 2923
Longest chain: 8
Shortest chain: 1
ach.magnificentpool.com [not blocked]
events3alt.adcolony.com [blocked]
ach.mine-vn.com [not blocked]
ach.nibirupool.com [not blocked]
www.amazonco.uk [blocked]
```

How to develop your code

There are three files, `Makefile, blocklist.txt and dnsblock.c`. Use the file `dnsblock.c` to write your program. `blocklist.txt` is a real blocklist file.

You must follow the following rules:

- You cannot change the name or parameters (name or count) of any function prototype given
- You cannot alter `main()`, `parseopts()` or `hash()` in any way
- You cannot change the typedef for struct node
- If given you must use any `printf()` or `fprintf()` format string
- You must check the return value of all functions you use in the body you write and print a descriptive error message to stderr

Part 2: Survey [1 point]

Complete this survey about your experiences this term. Double check your PID when entering to ensure you receive credit.

Style and Commenting

No points are explicitly given for style, but teaching staff won't be able to provide assistance or regrades unless code is readable. Please take a look at the following Style Guidelines

Submission and Grading

1. Submit your files to Gradescope under the assignment titled Homework4. You will submit the following file:

   `dnsblock.c`

2. After submitting, the autograder will run a few tests:
   a. Checks that all required file was submitted
   b. Checks that `dnsblock.c` compiles
   c. Runs some tests on `dnsblock.c`

Make sure to check the autograder output after submitting! We will be running additional tests after the deadline passes to determine your final grade.

The assignment will be graded out of 10 points, with 9 points allocated to the main `dnsblock` program. 1 point for completing the survey. Make sure your assignment compiles correctly on the ieng6 machines. **Any assignment that does not compile will receive 0 credit for part 1.**

References

[1] https://en.wikipedia.org/wiki/Domain_Name_System