COMP3331/9331 Computer Networks and Applications

Assignment for Term 2, 2024

Version 1.0

Due: 11:59am (noon) Friday, 26 July 2024 (Week 9)

Updates to the assignment, including any corrections and clarifications, will be posted on the course website. Please make sure that you check the course website regularly for updates.

Table of Contents

DNS Implementation

1. Goal and Learning Objectives

In this assignment, you will have the opportunity to implement a simplified version of DNS. Your applications are based on a client-server model consisting of one server and multiple clients, where:

1. A **client** can accept some user query, send it to the server, wait for a response, and upon receiving a response, display the results.
2. A **server** waits for client queries, and upon receiving a query, attempts to resolve it using resource records found in a master file, then replies with the appropriate response.

All client-server communication will be over **UDP** (TCP **cannot** be used for this assignment).

You will additionally submit a report.

On completing this assignment, you will gain sufficient expertise in the following skills:

1. A deeper understanding of how DNS works.
2. Socket programming for UDP transport protocol.
3. Designing an application layer protocol.

2. Master File

The master file is a text file that contains resource records (RRs) in text form. Each line contains one record. Any number of spaces act as a delimiter between the separate items that make up a record. The items that make up a record, in the order they appear on a line, are:

```
<domain-name>  <type>  <data>
```

Where:

- `domain-name`: is said to be the "owner" of the record. The labels in the domain name are expressed as character strings and are separated by dots.
    - All domain names will end with a dot.
    - All domain names will be 256 characters or less.
    - All domain names will be in lower case.
    - All labels within a domain name will start with a letter, end with a letter or digit, and have as interior characters only letters a-z, digits 0-9, and hyphens.

    - All labels will be 63 characters or less.
- `type`: specifies the type of the resource in the resource record, which will be one of the following:
    - `A`: a host address.
    - `CNAME`: identifies the canonical name of an alias.
    - `NS`: the authoritative name server for the domain.
- `data`: is the type-dependent data which describes the resource:
    - `A`: a 32-bit IP address expressed as four decimal numbers separated by dots and without any embedded spaces.
    - `CNAME`: a domain name, which follows the previously stated rules for domain names, and specifies the canonical or primary name for the owner. The owner name is an alias.
    - `NS`: a domain name, which follows the previously stated rules for domain names, and specifies a host which should be authoritative for the domain.

Resource records with the same `domain-name` and `type` form a resource set. The order of RRs in a set is not significant, and need not be preserved by the client or server.

You may assume that each resource record in the master file will be unique, but you should not assume any particular ordering of the master file.

You may assume that the master file contains at least one `NS` record for the root zone, and its corresponding `A` record, meaning the server can always refer the client to some other name server which will provide eventual access to an answer. You may also assume there will be no circular `CNAME` references, and no `NS` records that point to a `CNAME` record.

The master file will be named `master.txt`. It will be located in the working directory of the server, and it will follow the format and rules stated above.

Do not hard-code any resource records within your server program. The records must be read from the file when the server is executed. Failure to do so will likely result in your applications not passing any testing.

Do not hard-code any path to the master file, or assume any name other than `master.txt`, within your server program. The program must read the file `master.txt` from the current working directory. Failure to do so will likely result in your applications not passing any testing.

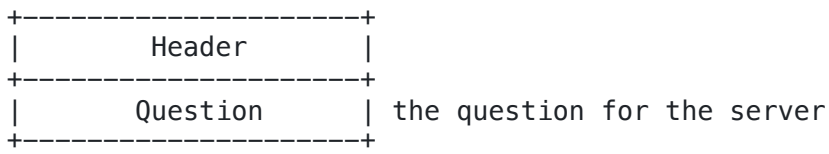
You may assume that the master file will contain no more than 256 records, there will be no blank lines, and there will be no leading or trailing whitespace on any line. Furthermore, the final record will have no trailing newline.

A sample master file is provided. A different master file will be used during marking. You may assume the file will be present, will have the same name, and will follow the same format and rules.

3. Message Format

You are to design the message format. Your design may utilise a single format for both queries and responses, or they may be distinct. However, conceptually the messages should include the information detailed below. You may assume that the resource records contained within any message, stripped of any whitespace between the fields, will not exceed 2048 bytes.

3.1. Client Query



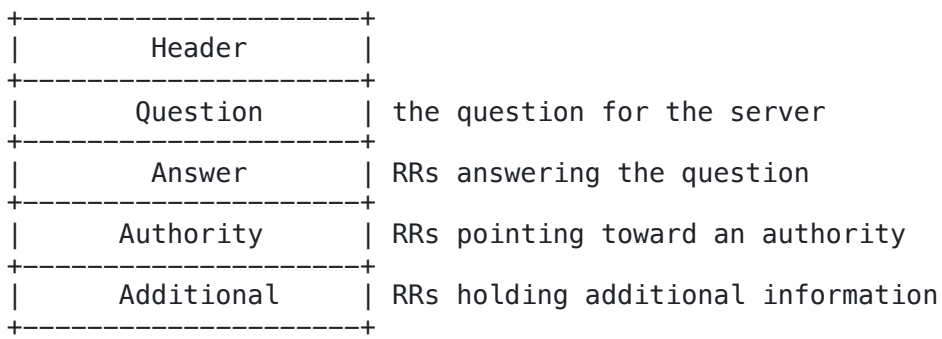
Minimally the header section should include:

- `qid`: A 16-bit unsigned integer, randomly generated by the client, as an identifier for the query. This identifier is copied into the corresponding response by the server.

The question section should include:

- `qtype`: The type of the query, and will be one of the types listed in Section 2. Master File.
- `qname`: The target domain name of the query, which will follow the rules detailed in Section 2. Master File

3.2. Server Response



Minimally the header section should include:

- `qid`: The 16-bit unsigned integer identifier of the query to which this is a response.

Similar to the `qid`, the question section duplicates the question section of the query.

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 section contains RRs which relate to the query, but are not strictly answers for the question. All RRs will follow the rules detailed in Section 2. Master File.

4. Client

4.1. File Name and Arguments

The main code for the client must be contained in one of the following files:

- `client.c`
- `Client.java`
- `client.py`

You are free to create additional helper files and name them as you wish.

The client should accept the following arguments:

1. `server_port`: The UDP port number on which the server is listening. This will be the same argument supplied to the server.
2. `qname`: The target domain name of the query. This will be well formed, as discussed in Section 2. Master File, will always be provided in lower case, and will always include a trailing `.` to represent the root domain.
3. `qtype`: The type of the query. This will be one of the types listed in Section 2. Master File, and will always be provided in upper case.
4. `timeout`: The duration in seconds, greater than 0 but no more than 15, which the client should wait for a response before considering it a failure and terminating.

Execution of your client should be initiated as follows:

C:

```
$ ./client server_port qname qtype timeout
```

Java:

```
$ java Client server_port qname qtype timeout
```

Python:

```
$ python3 client.py server_port qname qtype timeout
```

During testing, we will ensure that the command-line arguments provided are in the correct format. We will not test for erroneous arguments, missing arguments, etc. That said, it is good programming practice to check for such input errors.

4.2. Client Operation

Upon execution, the client should:

1. Construct a message that will carry the provided `qname` and `qtype` in its question section.
2. Assign a randomly generated, 16-bit unsigned integer as the `qid`.

3. Using an ephemeral port (i.e. one allocated by the operating system), send the message to the `server_port` of the localhost (`127.0.0.1`).
4. Wait up to `timeout` seconds for a response.
5. If received, print the response, otherwise, print an error message.
6. Terminate.

4.3. Client Output

You are free to nominate the output format, provided it is easily human readable, but minimally it must include:

- The `qid` carried in the response, as a decimal integer.
- A Question section heading, followed by the `qname` and `qtype` found in the response.
- For each of the Answer, Authority, and Additional sections:
    - A section heading, followed by the resource records contained in that section of the response.
    - You are free to choose whether or not a heading is printed out for any section that does not contain any records.

Important: Tutors will be relying on the output in order to mark your client. If it does not produce any output, then it cannot be marked. If the output is difficult to read, or some of the output is missing, then you should not expect full marks, even if your implementation is otherwise correct.

Please do not email course staff or post on the forum asking if your output is acceptable. This is not scalable for a course of this size. If you are unsure of your output, then you should follow the example output.

Examples are provided in Appendix G. Sample Interactions.

5. Server

5.1. File Name and Arguments

The main code for the server must be contained in one of the following files:

- `server.c`
- `Server.java`
- `server.py`

You are free to create additional helper files and name them as you wish.

The server should accept the following argument:

1. `server_port`: The UDP port number on which the server should listen for client queries. We recommend using a random port number between 49152 and 65535 (the dynamic port number range). This will be the same port argument supplied to the client.

Execution of your server should be initiated as follows:

C:

```
$ ./server server_port
```

Java:

```
$ java Server server_port
```

Python:

```
$ python3 server.py server_port
```

During testing, we will ensure that the command-line arguments provided are in the correct format. We will not test for erroneous arguments, missing arguments, etc. That said, it is good programming practice to check for such input errors.

5.2. Server Operation

Upon execution, the server should:

1. Read in the master file, and initialise an in-memory cache with these records.
    a. You are free to design the cache internals, with the help of any general purpose (i.e. non-DNS) standard libraries, as you see fit.
2. Bind to the `server_port` of the loopback interface (`127.0.0.1`).
    a. During testing the server and all clients will be executed on the same host.
3. Loop forever, listening for queries. Upon receiving a query, the server should:
    a. Delay processing the query for a random amount of time of 0, 1, 2, 3, or 4 seconds. Meanwhile the server should still be able to listen for (and similarly delay/process) other queries. This will allow us to assess how well your server can multiplex queries from multiple clients.
    b. Once the delay period has elapsed for a given query, construct the appropriate response, utilising its cache.
    c. Send the response back to the client.

The server needs to support multiple clients simultaneously. A robust way to achieve this is to use multithreading. In this approach, you might have a main thread to listen for new queries. When a query is received the server might spawn a new thread, which will sleep for the delay duration, process the query, send a response, and then terminate. However, we do not mandate any specifics. You may elect to solve this some other way. Regardless of your approach, you should not assume any arbitrary limits on the number and frequency of clients.

You may also note that during testing the server will always be started before any client requests are made, the server will not be manually terminated, and should the server crash, it will be restarted.

5.3. Query Processing

1. Attempt to match the `qname`.
    a. If the whole of `qname` is matched, we have found the node.

If the data at the node is a `CNAME`, and `qtype` doesn't match `CNAME`, copy the `CNAME` RR into the answer section of the response, change `qname` to the canonical name found in the `CNAME` RR, and start again. Note, in this scenario the ordering of the `CNAME` records in the answer section should follow the order in which they're added, which should also be reflected in the client output.

Otherwise, copy all RRs which match `qtype` into the answer section, and go to step 2.

b. If the match fails, we have a referral.

Find the closest ancestor zone to `qname` for which there are known name servers. Copy all `NS` RRs for the zone into the authority section. For each name server, copy any known `A` records into the additional section. Go to step 2.

2. Send the response.

You may assume that all queries are resolvable, via an answer and/or a referral.

5.4. Server Output

You are free to nominate the output format, provided it is easily human readable, but it must log to the terminal all messages sent and received, and minimally include:

- A timestamp of when the message was sent or received, with at least millisecond precision.
- Whether the message was sent or received.
- The client port number.
- The `qid` of the message, as a decimal integer.
- The `qname` and `qtype` carried in the message.
- If the message is a query received, the period of delay until the query will be processed.
    - Note, this should be printed when the query is received, not when the query is processed.

A general template may look like:

```
<timestamp> <snd|rcv> <client_port>: <qid> <qname> <qtype> (delay: <delay>s)
```

Important: Tutors will be relying on the output in order to mark your server. If it does not produce any output, then it cannot be marked. If the output is difficult to read, or some of the output is missing, then you should not expect full marks, even if your implementation is otherwise correct.

Please do not email course staff or post on the forum asking if your output is acceptable. This is not scalable for a course of this size. If you are unsure of your output, then you should follow the example output.

Examples are provided in Appendix G. Sample Interactions.

6. Report

You should submit a small report (no more than 2 pages) named `report.pdf`. This should include:

- Details of which language you have used (e.g., C) and the organisation of your code (`Makefile`, directories if any, etc.).
- The overall program design and any data structure design, for example, any data structures used to represent resource records, application messages, and the cache.
- Known limitations, for example, if your program does not work under certain circumstances, then report those circumstances.
- Also indicate any segments of code that you have borrowed from the Web or other sources.

Appendices

A. Languages and Permitted Features

You are free to use C, Java, or Python. Please choose a language with which you are comfortable.

Your submission will be tested on the CSE servers. For this reason, it is **critical** that your code can compile and run in that environment, using the standard system installations. This is especially important if you plan to develop and test your code on your personal computer.

For your reference, the CSE machines currently support the following:

- C: gcc v12.2
- Java: openjdk v17.0
- Python: python3 v3.11

If your code does not run on the CSE servers, then it cannot be marked, and it will be awarded ZERO.

You **must** implement all functionality yourself. For example, you **must not**:

- Utilise any "ready-made" server libraries.
- Utilise any DNS libraries.
- Make standard library calls to functions like `gethostbyname` or `gethostbyaddr`.
- Send queries to any DNS resolver or server.
- Invoke similar tools, such as `dig` or `nslookup`.

Attempting to circumvent the intent of the assignment will similarly result in a mark of ZERO.

If you are unsure about anything, please check with the course staff on the forum.

B. Additional Notes

- This is **not** a group assignment. You are expected to work on this individually.
- **Sample Code**: We will provide sample code on the assignment page in all 3 programming languages. You are not required to use it, but you are welcome to adapt it as you see fit, to help get started.
- **Programming Tutorial**: We will run programming tutorials, for all 3 programming languages, during regular lab times in Week 7, to get students started with programming some of the building blocks of the assignment. A schedule for these sessions will be announced no later than Week 6.
- **Assignment Help Sessions**: We will run additional consultations, for all 3 programming languages, from Week 7 on, to assist you with assignment related questions. A schedule will be posted on the assignment page of the course website no later than Week 6. Please note, these sessions are not a forum for tutors to debug your code.
- **Backup and Versioning**: We strongly recommend you back-up your programs frequently. CSE backs up all user accounts nightly. If you are developing code on your personal machine, it is strongly recommended that you undertake daily backups. We also recommend using a

good versioning system so that you can roll back and recover from any inadvertent changes. There are many services available for this, which are easy to use. We will **not** entertain any requests for special consideration due to issues related to computer failure, lost files, etc.

- **Debugging**: When implementing a complex assignment such as this, there are bound to be errors in your code. We strongly encourage that you follow a systematic approach to debugging. If you are using an IDE for development, then it is bound to have debugging functionalities. Alternatively, you could use a command line debugger such as `pbd` (Python), `jdb` (Java) or `gdb` (C). Use one of these tools to step through your code, create break points, observe the values of relevant variables and messages exchanged, etc. Proceed step by step, check and eliminate the possible causes until you find the underlying issue. Note that, we won't be able to debug your code on the course forum or in the help sessions.
- It is imperative that you rigorously test your code to ensure that all possible (and logical) interactions can be correctly executed. **Test, test, and test**.
- You are encouraged to use the course discussion forum to ask questions and to discuss different approaches to solve the problem. However, you **must not** post your solution or any code fragments on the forum, or on any other public medium.

C. Submission

Please ensure that you use the mandated file names of `report.pdf` and for the entry point of each application. These are:

Language	Client	Server
C	`client.c`	`server.c`
Java	`Client.java`	`Server.java`
Python	`client.py`	`server.py`

If you are using C or Java, then you **must** additionally submit a `Makefile`. This is because we need to know how to resolve any dependencies. After running make, we should have the following executable files:

Language	Client	Server
C	`client`	`server`
Java	`Client.class`	`Server.class`

If your submission does not rely on a directory structure, then you may submit the files directly. For example, assuming the relevant files are in the current working directory:

```
$ give cs3331 assign client.c server.c Makefile report.pdf
```

However, if your submission does rely on some directory structure, then you **must** first `tar` the parent directory. For example, assume a directory `assign` contains all the relevant files and sub-directories. Open a terminal and navigate to the parent directory of `assign`, then:

```
$ tar -cvf assign.tar assign
$ give cs3331 assign assign.tar
```

Upon running `give`, ensure that your submission is accepted. You may submit often. Only your last submission will be marked.

Emailing your code to course staff will not be considered as a submission.

Submitting the wrong files, failing to submit certain files, failing to complete the submission process, or simply failing to submit, will not be considered as grounds for re-assessment.

If you wish to validate your submission, you may execute:

```
$ 3331 classrun -check assign  # show submission status
$ 3331 classrun -fetch assign  # fetch most recent submission
```

Important: It is your responsibility to ensure that your submission is accepted, and that your submission is what you intend to have assessed. **No exceptions**.

D. Late Submission Policy

Late submissions will incur a 5% per day penalty, for up to 5 days, calculated on the achieved mark. Each day starts from the deadline and accrues every 24 hours.

For example, an assignment otherwise assessed as 12/20, submitted 49 hours late, will incur a 3 day x 5% = 15% penalty, applied to 12, and be awarded 12 x 0.85 = 10.2/20.

Submissions after 5 days from the deadline will not be accepted unless an extension has been granted, as detailed in Appendix E. Special Consideration and Equitable Learning Services.

E. Special Consideration and Equitable Learning Services

Applications for Special Consideration **must** be submitted to the university via the Special Consideration portal. Course staff do **not** accept or approve special consideraton requests.

Students who are registered with Equitable Learning Services **must** email cs3331@cse.unsw.edu.au to request any adjustments based on their Equitable Learning Plan.

Any requested and approved extensions will defer late penalties and submission closure. For example, a student who has been approved for a 3 day extension, will not incur any late penalties until 3 days after the standard deadline, and will be able to submit up to 8 days after the standard deadline.

F. Plagiarism

You are to write all the code for this assignment yourself. All source code is subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous terms. In addition, each submission will be checked against all other submissions of the current term. Do not post this assignment on forums where you can pay programmers to write code for you. We will be monitoring such forums. Please note that we take this matter quite seriously. The LiC will decide on the

appropriate penalty for detected cases of plagiarism. The most likely penalty will be to reduce the assignment mark to ZERO.

We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide or accept any programming code in writing, as this is likely to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the code. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over.

It is OK to borrow short snippets of sample socket code out on the Web and in books. You **must**, however, acknowledge the source of any borrowed code. This means providing a reference to a book or a URL (as comments) where the code appears. Also indicate in your report the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

Generative AI Tools: It is prohibited to use any software or service to search for or generate information or answers. If its use is detected, it will be regarded as serious academic misconduct and subject to the standard penalties, which may include 00FL, suspension and exclusion.

G. Sample Interactions

Note, these interactions are based on the provided sample master file. You should be familiar with the resource records contained in the master file in order to understand these interactions.

For these examples, we assume the implementation is in Python, but the arguments would be exactly similar for C or Java. The server is long-running, and has been executed with:

```
$ python3 server.py 54321
```

G.1. example.com. A

Client execution:

```
$ python3 client.py 54321 example.com. A 5
ID: 17564

QUESTION SECTION:
example.com.  A

ANSWER SECTION:
example.com.  A  93.184.215.14
```

Server log:

```
2024-05-21 19:20:31.750 rcv 62370: 17564 example.com. A (delay: 3s)
2024-05-21 19:20:34.756 snd 62370: 17564 example.com. A
```

G.2. example.com. A (timeout)

Client execution:

```
$ python3 client.py 54321 example.com. A 1
timed out
```

Server log:

```
2024-05-21 19:20:34.815 rcv 60780:  1885 example.com. A (delay: 2s)
2024-05-21 19:20:36.822 snd 60780:  1885 example.com. A
```

G.3. bar.example.com. CNAME

Client execution:

```
$ python3 client.py 54321 bar.example.com. CNAME 5
ID: 8006

QUESTION SECTION:
bar.example.com.  CNAME

ANSWER SECTION:
bar.example.com.  CNAME  foobar.example.com.
```

Server log:

```
2024-05-21 19:20:35.875 rcv 61445:  8006 bar.example.com. CNAME (delay: 3s)
2024-05-21 19:20:38.881 snd 61445:  8006 bar.example.com. CNAME
```

G.4. . NS

Client execution:

```
$ python3 client.py 54321 . NS 5
ID: 26888

QUESTION SECTION:
.  NS

ANSWER SECTION:
.  NS  b.root-servers.net.
.  NS  a.root-servers.net.
```

Server log:

```
2024-05-21 19:20:38.942 rcv 56853: 26888 . NS (delay: 3s)
2024-05-21 19:20:41.948 snd 56853: 26888 . NS
```

G.5. bar.example.com. A

Client execution:

```
$ python3 client.py 54321 bar.example.com. A 5
ID: 266

QUESTION SECTION:
bar.example.com.     A

ANSWER SECTION:
bar.example.com.     CNAME  foobar.example.com.
foobar.example.com.  A      192.0.2.23
foobar.example.com.  A      192.0.2.24
```

Server log:

```
2024-05-21 19:20:42.006 rcv 56483:   266 bar.example.com. A (delay: 4s)
2024-05-21 19:20:46.012 snd 56483:   266 bar.example.com. A
```

G.6. foo.example.com. A

Client execution:

```
$ python3 client.py 54321 foo.example.com. A 5
ID: 2735

QUESTION SECTION:
foo.example.com.     A

ANSWER SECTION:
foo.example.com.     CNAME  bar.example.com.
bar.example.com.     CNAME  foobar.example.com.
foobar.example.com.  A      192.0.2.23
foobar.example.com.  A      192.0.2.24
```

Server log:

```
2024-05-21 19:20:46.071 rcv 54940:  2735 foo.example.com. A (delay: 3s)
2024-05-21 19:20:49.076 snd 54940:  2735 foo.example.com. A
```

G.7. example.org. A

Client execution:

```
$ python3 client.py 54321 example.org. A 5
ID: 51716

QUESTION SECTION:
example.org.         A

AUTHORITY SECTION:
.                    NS  b.root-servers.net.
.                    NS  a.root-servers.net.

ADDITIONAL SECTION:
a.root-servers.net.  A   198.41.0.4
```

Server log:

```
2024-05-21 19:20:49.086 rcv 60942: 51716 example.org. A (delay: 0s)
2024-05-21 19:20:49.087 snd 60942: 51716 example.org. A
```

G.8. example.org. CNAME

Client execution:

```
$ python3 client.py 54321 example.org. CNAME 5
ID: 42020

QUESTION SECTION:
example.org.         CNAME

AUTHORITY SECTION:
.                    NS     b.root-servers.net.
.                    NS     a.root-servers.net.

ADDITIONAL SECTION:
a.root-servers.net.  A      198.41.0.4
```

Server log:

```
2024-05-21 19:20:49.161 rcv 57836: 42020 example.org. CNAME (delay: 1s)
2024-05-21 19:20:50.167 snd 57836: 42020 example.org. CNAME
```

G.9. example.org. NS

Client execution:

```
$ python3 client.py 54321 example.org. NS 5
ID: 6775

QUESTION SECTION:
example.org.         NS

AUTHORITY SECTION:
.                    NS  b.root-servers.net.
.                    NS  a.root-servers.net.

ADDITIONAL SECTION:
a.root-servers.net.  A   198.41.0.4
```

Server log:

```
2024-05-21 19:20:50.217 rcv 60553:  6775 example.org. NS (delay: 2s)
2024-05-21 19:20:52.222 snd 60553:  6775 example.org. NS
```

G.10. www.metalhead.com. A

Client execution:

```
$ python3 client.py 54321 www.metalhead.com. A 5
ID: 61240

QUESTION SECTION:
www.metalhead.com.   A

ANSWER SECTION:
www.metalhead.com.   CNAME  metalhead.com.

AUTHORITY SECTION:
com.                 NS     d.gtld-servers.net.

ADDITIONAL SECTION:
d.gtld-servers.net.  A      192.31.80.30
```

Server log:

```
2024-05-21 19:20:52.281 rcv 53718: 61240 www.metalhead.com. A (delay: 0s)
2024-05-21 19:20:52.281 snd 53718: 61240 www.metalhead.com. A
```

G.11. Multiple Clients

This sample interaction demonstrates all previous examples, but where the server needs to process multiple queries concurrently. The client output is as previously given, just with different query IDs (as reflected in the server log).

Server log:

```
2024-05-21 19:27:48.067 rcv 53465: 14776 example.com. A (delay: 0s)
2024-05-21 19:27:48.067 snd 53465: 14776 example.com. A
2024-05-21 19:27:48.072 rcv 51203: 57250 example.com. A (delay: 2s)
2024-05-21 19:27:48.084 rcv 51322: 26495 bar.example.com. CNAME (delay: 0s)
2024-05-21 19:27:48.084 snd 51322: 26495 bar.example.com. CNAME
2024-05-21 19:27:48.098 rcv 60681: 29599 . NS (delay: 4s)
2024-05-21 19:27:48.110 rcv 64756:  1544 bar.example.com. A (delay: 0s)
2024-05-21 19:27:48.110 snd 64756:  1544 bar.example.com. A
2024-05-21 19:27:48.121 rcv 61511:  8551 foo.example.com. A (delay: 1s)
2024-05-21 19:27:48.136 rcv 60198:  6165 example.org. A (delay: 1s)
2024-05-21 19:27:48.149 rcv 60197: 61858 example.org. CNAME (delay: 0s)
2024-05-21 19:27:48.149 snd 60197: 61858 example.org. CNAME
2024-05-21 19:27:48.165 rcv 61165: 13006 example.org. NS (delay: 1s)
2024-05-21 19:27:48.180 rcv 51479: 35102 www.metalhead.com. A (delay: 3s)
2024-05-21 19:27:49.126 snd 61511:  8551 foo.example.com. A
2024-05-21 19:27:49.141 snd 60198:  6165 example.org. A
2024-05-21 19:27:49.171 snd 61165: 13006 example.org. NS
2024-05-21 19:27:50.078 snd 51203: 57250 example.com. A
2024-05-21 19:27:51.186 snd 51479: 35102 www.metalhead.com. A
2024-05-21 19:27:52.103 snd 60681: 29599 . NS
```

H. Marking Policy

H.1. Preliminary Checks

We will perform both automated and manual checks to assess:

1. That the submitted code is original. See Appendix F. Plagiarism.
2. That the submitted code can compile and run in the standard CSE lab environment, and only uses permitted features. See Appendix A. Languages and Permitted Features.
3. That the submitted code utilises UDP.

Failing any of the above will likely result in a mark of 0, and in the case of plagiarism, possibly further disciplinary action.

H.2. Report: 1 mark

As described in Section 6. Report. We will verify that the description in your report confirms with the actual implementations in your programs.

H.3. Properly documented and commented code: 1 mark

No particular style guide is mandated, but you might consider following:

- C Coding Style Guide
- Google Java Style Guide
- PEP 8 - Style Guide for Python Code

H.4. Single Client: 12 marks

Please note, scripts may be used during marking to automate the execution of your applications (for both single client and multiple clients), but all behaviour will be assessed manually by a tutor.

Using a different master file to the sample master file that's provided, we will first execute your server. We will then execute various queries, similar to those in Appendix G. Sample Interactions, running one client at a time. For each, we will consider query names at various levels of the DNS hierarchy.

In the first scenario, we will execute queries that can resolve to a single answer without any referrals or query restarts. That is, there will be no need to populate the authority or additional sections of the response, and the server will never encounter an alias that would require the query to restart.

We will test with the following query types:

- `A` (1 mark)
- `CNAME` (1 mark)
- `NS` (1 mark)

The second scenario will be similar to the first, however there may be multiple answers.

We will test with the following query types:

- `A` (1 mark)
- `NS` (1 mark)

In the third scenario, we will execute queries that result in a referral, without any answers or query restarts.

We will test with the following query types:

- `A` (1 mark)
- `CNAME` (1 mark)
- `NS` (1 mark)

The referrals may include any number of expected records in the authority and additional sections.

In the fourth scenario, we will execute queries that may encounter one or more aliases and require query restart, but will not require any referrals.

We will test with the following query type:

- `A` (2 marks)

In the fifth scenario, we will execute queries that require both query restart and referrals.

We will test with the following query type:

- `A` (1 mark)

Finally, we will test:

- Client timeout (1 mark)

Using various timeout values.

Throughout testing we will also assess whether the general requirements of the assignment have been met. For instance:

- Whether the server is binding to the port given as a command-line argument.
- Whether the client is using an ephemeral port.
- Whether the query ID is randomly generated.
- Whether the output is complete and easily readable.

Failure to meet any of the general requirements may result in penalties.

However, we will not assess the server's query processing delay during this stage of the marking.

H.5. Multiple Clients: 6 marks

Using a different master file to the sample master file that's provided, we will first execute your server. We will then execute various queries, using multiple clients simultaneously. For each, we will consider query names at various levels of the DNS hierarchy.

We will assess the server processing delay to ensure it's random, in the correct range, and accurate. Without evidence of this random delay and the server being able to receive and delay/process multiple queries concurrently, it will not be possible to assess this aspect of your submission.

Also note, the server log showing some minor variation (e.g. 10 milliseconds), in addition to the random delay, between delaying a query and sending a response, is expected and acceptable.

The various scenarios we will test are:

- The same as scenario 1 for the single client, but running all types simultaneously (2 marks)
- The same as scenario 3 for the single client, but running all types simultaneously (2 marks)
- Any combination of scenarios from the single client, run simultaneously (2 marks)