Computer Networks Lab Report- Assignment 8

TITLE:

Name: Debarghya Maitra
Class: BCSE 3 rd Year
Group: A3
Submission Date: 11/11/2022

PROBLEM STATEMENT: Implement any 2 of the following protocols using TCP/UDP sockets: FTP, DNS, Telnet.

DESIGN:

1) For the implementation of DNS or Domain Name System server the following procedure has been adopted:

a. **entries.json**: Contains all the records presently stored in the server.

b. **lookup.go**: Deals with JSON parsing, reading previous records and storing new records, which is part of the whole package.

c. **main.go**: Handles all DNS requests and tries to resolve a domain name to respective IP addresses with the following process:

i. DNS Header is maintained in the following struct:

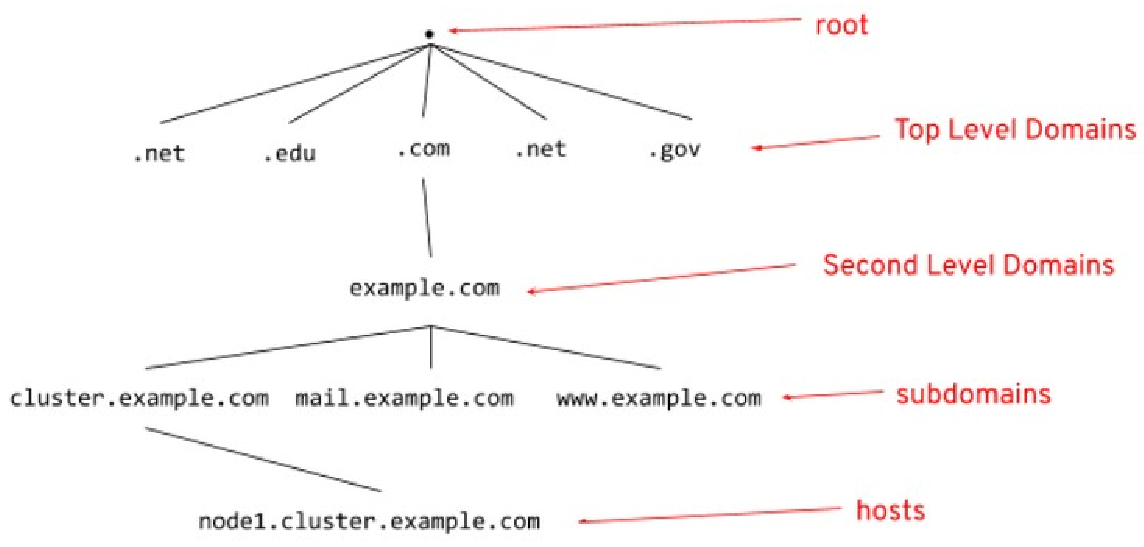
```
type DNSHeader struct {
	TransactionID  uint16
	Flags          uint16
	NumQuestions   uint16
	NumAnswers     uint16
	NumAuthorities uint16
	NumAdditionals uint16
}
```

ii. DNS Resource Record is maintained by the following struct:

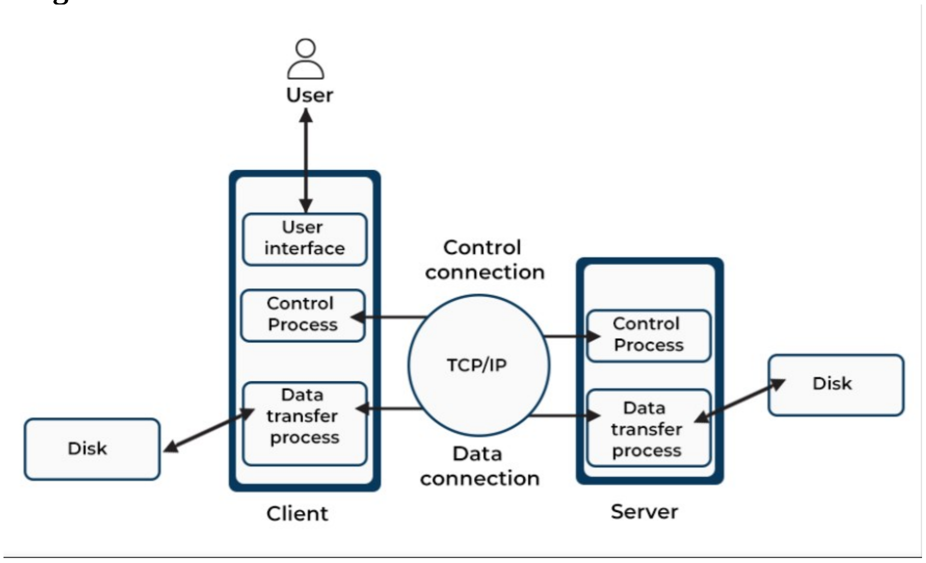
```
type DNSResourceRecord struct {
	DomainName         string
	Type               uint16
	Class              uint16
	TimeToLive         uint32
	ResourceDataLength uint16
	ResourceData       []byte
}
```

iii. The main function invokes a new go routine when a new request arrives and is handled separately,
Next the particular record is searched iteratively in the json file according to the root DNS i.e com, org, in, etc.
If it can resolve the IP of the particular domain, it returns immediately else it further searches into the system’s DNS server to resolve the domain.
Finally it returns the resource record according to the packet to particular client.

The following image shows how the server works as a whole:



2) For the implementation of FTP the following design pattern has been followed:
a) **creds**: The creds, or credentials package deals with the encrypted usernames and passwords stored in credentials.json. Stores and extracts the passwords.
b) **client** : The client package deals with handling the FTP client separately:
i) The folder "ftpfiles" maintains the default local directory for the client from where file exanges/downloading occurs.
ii) The main function invokes a go routine first to verify if a client is valid by checking username and password.
iii) Then it invokes two more go routines if it gets validated: One maintains a telnet type connection to parse remote commands to be executed on the server. The other one is mainly responsible for dealing with the file transfer(both upload and download). Currently the following commands are supported: pwd, ls, cat, del, cd, get, put.
c) **server**: The server package maintains the folder "ftp" to deal with the file transfers. Quite similar to the client, the server also invokes 2 go routines to maintain the FTP. One for parsing commands and another for file transfers.
d) One **important security mitigation** that has been adopted is that **a client can't exit the ftp directory by using cd command.**



IMPLEMENTATION

1) DNS client handler function:

```
func handleDNSClient(requestBytes []byte, serverConn *net.UDPConn,
clientAddr *net.UDPAddr) {
	var requestBuffer = bytes.NewBuffer(requestBytes)
	var queryHeader DNSHeader
	var queryResourceRecords []DNSResourceRecord
	err := binary.Read(requestBuffer, binary.BigEndian, &queryHeader)
	if err != nil {
		log.Println("Error decoding header: ", err.Error())
	}
	queryResourceRecords = make([]DNSResourceRecord,
queryHeader.NumQuestions)
	for idx, _ := range queryResourceRecords {
		queryResourceRecords[idx].DomainName, err =
readDomainName(requestBuffer)
		if err != nil {
			log.Println("Error decoding label: ", err.Error())
		}
		queryResourceRecords[idx].Type =
binary.BigEndian.Uint16(requestBuffer.Next(2))
		queryResourceRecords[idx].Class =
binary.BigEndian.Uint16(requestBuffer.Next(2))
	}
	var answerResourceRecords = make([]DNSResourceRecord, 0)
	var authorityResourceRecords = make([]DNSResourceRecord, 0)
	var additionalResourceRecords = make([]DNSResourceRecord, 0)
	for _, queryResourceRecord := range queryResourceRecords {
		newAnswerRR, newAuthorityRR, newAdditionalRR :=
dbLookup(queryResourceRecord)
		answerResourceRecords = append(answerResourceRecords,
newAnswerRR...)
		authorityResourceRecords = append(authorityResourceRecords,
newAuthorityRR...)
		additionalResourceRecords = append(additionalResourceRecords,
newAdditionalRR...)
	}
	var responseBuffer = new(bytes.Buffer)
	var responseHeader = DNSHeader{
		TransactionID:  queryHeader.TransactionID,
		Flags:          FlagResponse,
		NumQuestions:   queryHeader.NumQuestions,
		NumAnswers:     uint16(len(answerResourceRecords)),
		NumAuthorities: uint16(len(authorityResourceRecords)),
		NumAdditionals: uint16(len(additionalResourceRecords)),
	}
	err = Write(responseBuffer, &responseHeader)
	if err != nil {
		log.Println("Error writing to buffer: ", err.Error())
	}
	for _, queryResourceRecord := range queryResourceRecords {
		err = writeDomainName(responseBuffer,
queryResourceRecord.DomainName)
		if err != nil {
			log.Println("Error writing to buffer: ", err.Error())
		}
		Write(responseBuffer, queryResourceRecord.Type)
		Write(responseBuffer, queryResourceRecord.Class)
	}
	for _, answerResourceRecord := range answerResourceRecords {
		err = writeDomainName(responseBuffer,
answerResourceRecord.DomainName)
		if err != nil {
			log.Println("Error writing to buffer: ", err.Error())
		}
		Write(responseBuffer, answerResourceRecord.Type)
		Write(responseBuffer, answerResourceRecord.Class)
		Write(responseBuffer, answerResourceRecord.TimeToLive)
		Write(responseBuffer, answerResourceRecord.ResourceDataLength)
		Write(responseBuffer, answerResourceRecord.ResourceData)
	}
	for _, authorityResourceRecord := range authorityResourceRecords {
		err = writeDomainName(responseBuffer,
authorityResourceRecord.DomainName)
		if err != nil {
			log.Println("Error writing to buffer: ", err.Error())
		}
		Write(responseBuffer, authorityResourceRecord.Type)
		Write(responseBuffer, authorityResourceRecord.Class)
		Write(responseBuffer, authorityResourceRecord.TimeToLive)
		Write(responseBuffer, authorityResourceRecord.ResourceDataLength)
		Write(responseBuffer, authorityResourceRecord.ResourceData)
	}
	for _, additionalResourceRecord := range additionalResourceRecords {
		err = writeDomainName(responseBuffer,
additionalResourceRecord.DomainName)
		if err != nil {
			log.Println("Error writing to buffer: ", err.Error())
		}
		Write(responseBuffer, additionalResourceRecord.Type)
		Write(responseBuffer, additionalResourceRecord.Class)
		Write(responseBuffer, additionalResourceRecord.TimeToLive)
		Write(responseBuffer,
additionalResourceRecord.ResourceDataLength)
		Write(responseBuffer, additionalResourceRecord.ResourceData)
	}
	serverConn.WriteToUDP(responseBuffer.Bytes(), clientAddr)
}
```

2) DNS lookup function:

```
func dbLookup(queryResourceRecord DNSResourceRecord) ([]DNSResourceRecord,
[]DNSResourceRecord, []DNSResourceRecord) {
	var answerResourceRecords = make([]DNSResourceRecord, 0)
	var authorityResourceRecords = make([]DNSResourceRecord, 0)
	var additionalResourceRecords = make([]DNSResourceRecord, 0)
	names, err := GetNames()
	if err != nil {
	return answerResourceRecords, authorityResourceRecords,
		additionalResourceRecords
	}
	if queryResourceRecord.Type != TypeA||queryResourceRecord.Class !=
ClassINET {
		return answerResourceRecords, authorityResourceRecords,
additionalResourceRecords
	}
	for _, name := range names {
	if strings.Contains(queryResourceRecord.DomainName,name.DomainName) {
			log.Println(queryResourceRecord.DomainName, "resolved to",
name.Address)
```

```
                answerResourceRecords = append(answerResourceRecords,
DNSResourceRecord{
                     DomainName:         name.DomainName,
                     Type:               TypeA,
                     Class:              ClassINET,
                     TimeToLive:         31337,
                     ResourceData:       name.Address[12:16],
                     ResourceDataLength: 4,
                })
           }
      }
      if len(answerResourceRecords) == 0 {
           ips, err := net.LookupIP(queryResourceRecord.DomainName)
           if err != nil {
                log.Panic("Could not get IPs:", err)
           }
           for _, ip := range ips {
           log.Println(queryResourceRecord.DomainName,"resolved to", ip)
                if ip.To4() != nil {
                     answerResourceRecords = append(answerResourceRecords,
DNSResourceRecord{
                          DomainName:      queryResourceRecord.DomainName,
                          Type:               TypeA,
                          Class:              ClassINET,
                          TimeToLive:         31337,
                          ResourceData:       ip[12:16],
                          ResourceDataLength: 4,
                     })
                     WriteNames(Record{
                          DomainName:    queryResourceRecord.DomainName,
                          Address: ip,
                     })
                }
           }
      }
      return answerResourceRecords, authorityResourceRecords,
additionalResourceRecords
}
```

3) FTP server handler function:

```
func serverHandler(conn net.Conn)  {
      defer conn.Close()
      authenticateClient(conn)
      buffer := make([]byte, BUFFSIZE)
      for{
           n, _ := conn.Read(buffer)
           cmd := string(buffer[:n])
           cmdList := strings.Split(cmd, " ")
           switch strings.ToLower(cmdList[0]){
           case "put":
                log.Println("Put File", cmdList[1])
                n, _ := conn.Read(buffer)
                fileSize, err := strconv.ParseInt(string(buffer[:n]),10,64)
                log.Println(fileSize)
                if err != nil || fileSize == -1{
                     log.Println("Error while parsing file")
                     continue
                }
                putFile(conn, cmdList[1], fileSize)
           case "get":
```

```
				log.Println("Get File", cmdList[1])
				getFile(conn, cmdList[1])
			case "ls":
				log.Println("ls")
				readDir(conn)
			case "pwd":
				log.Println("pwd")
				conn.Write([]byte(ROOTDIR))
			case "cd":
				chDir(conn, cmdList[1])
			case "cat":
				showFileContents(conn, cmdList[1])
			case "del":
				delDir(conn, cmdList[1])
			case "help":
				log.Println("Help Menu Display")
			case "quit":
				log.Println("Client Closed the Connection")
				return
			case "":
				log.Println("Error in buffer read!")
				return
			default:
				log.Println("Invalid command", cmd)
				continue
			}
		}
}
```

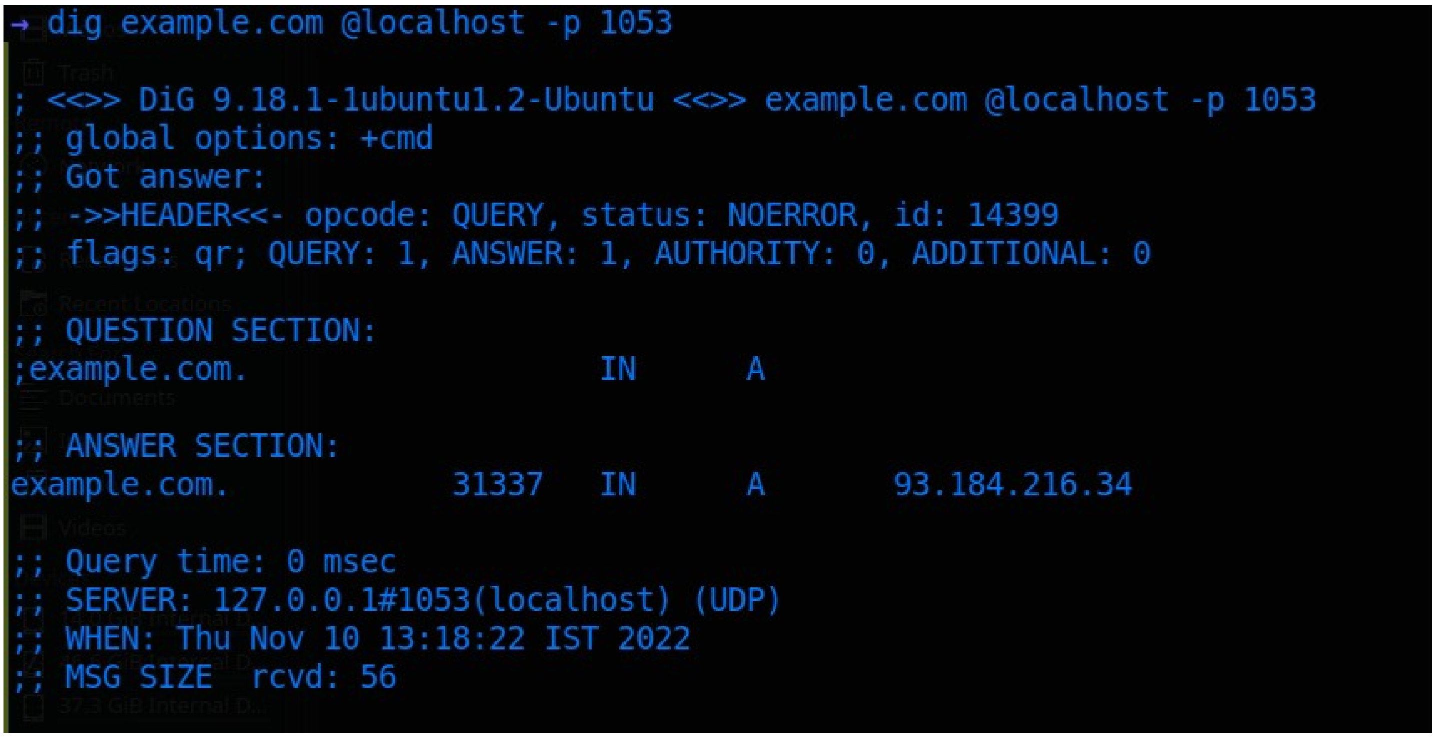
4) FTP client handler function:

```
func clientHandler(conn net.Conn)  {
	stdreader := bufio.NewReader(os.Stdin)
	buffer := make([]byte, BUFFSIZE)
	for{
		fmt.Printf("ftp> ")
		cmd, _  := stdreader.ReadString('\n')
		cmd = strings.Trim(cmd, "\n")
		cmdList := strings.Split(cmd, " ")
		switch strings.ToLower(cmdList[0]) {
		case "get":
			if len(cmdList) == 1 {
			fmt.Println("Invalid command\nType 'help' for more info.")
				continue
			}
			conn.Write([]byte(cmd))
			n, _ := conn.Read(buffer)
			fileSize , err := strconv.ParseInt(string(buffer[:n]), 10, 64)
			if err != nil{
				fmt.Println("ERROR: ", string(buffer[:n]))
				continue
			}
			saveFile(conn, cmdList[1], fileSize)
		case "put":
			if len(cmdList) == 1 {
			fmt.Println("Invalid command\nType 'help' for more info.")
				continue
			}
			conn.Write([]byte(cmd))
			sendFile(conn, cmdList[1])
		case "del":
			if len(cmdList) == 1 {
			fmt.Println("Invalid command\nType 'help' for more info.")
```

```
                        continue
                  }
                  conn.Write([]byte(cmd))
                  n, _ := conn.Read(buffer)
                  fmt.Println(string(buffer[:n]))
            case "cd":
                  if len(cmdList) == 1 {
                  fmt.Println("Invalid command\nType 'help' for more info.")
                        continue
                  }
                  conn.Write([]byte(cmd))
                  n, _ := conn.Read(buffer)
                  fmt.Println(string(buffer[:n]))
            case "pwd":
                  conn.Write([]byte(cmd))
                  n, _ := conn.Read(buffer)
                  fmt.Println(string(buffer[:n]))
            case "ls":
                  conn.Write([]byte(cmd))
                  n, _ := conn.Read(buffer)
                  fmt.Println(string(buffer[:n]))
            case "cat":
                  if len(cmdList) == 1 {
                  fmt.Println("Invalid command\nType 'help' for more info.")
                        continue
                  }
                  conn.Write([]byte(cmd))
                  tmp := make([]byte, BUFFSIZE)
            for {
                  n, err := conn.Read(tmp)
                  if err != nil {
                        if err != io.EOF {
                        fmt.Println("Read Error:", err)
                        }
                        break
                        }
                        if string(tmp[:n]) == "EOF"{
                              break
                        }
                  fmt.Println(string(tmp[:n]))
            }
            case "quit":
                  fmt.Println("Logging out\nGoodbye!")
                  conn.Write([]byte(cmd))
                  return
            case "help":
                  conn.Write([]byte(cmd))
                  fmt.Print("Supported Commands:\n\nget <filename>:\tDownload
remote file\nput <filename>:\tUpload local file\ndel <filename/dir>:\tDelete
remote dir/file\ncd <dirname>:\tChange directory\nls:\tList all
file(s)/directories in current remote directory\npwd:\tGet the present working
directory name\ncat <filename>:\tDisplay file contents\nquit:\tClose the
connection\nhelp:\tShow this menu.\n\n")
            case "":
                  continue
            default:
                  fmt.Println("Invalid command! To know all supported commands
use 'help'")
            }
      }
}
```
5) To test the DNS server, the UNIX inbuilt “dig” command has been used.

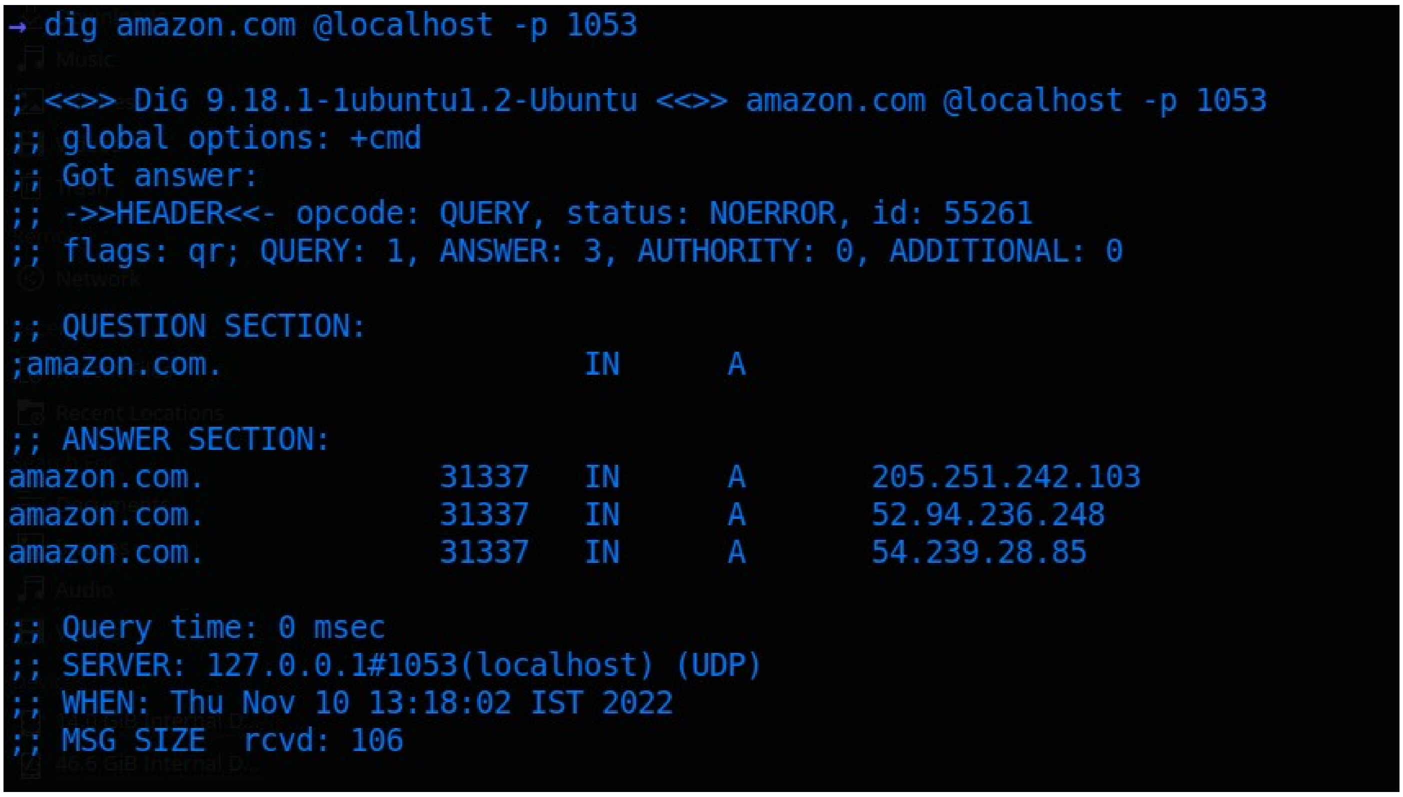
RESULTS

1) DNS Results: To test the DNS server we’re using the dig command provided by the GNU-utils package.



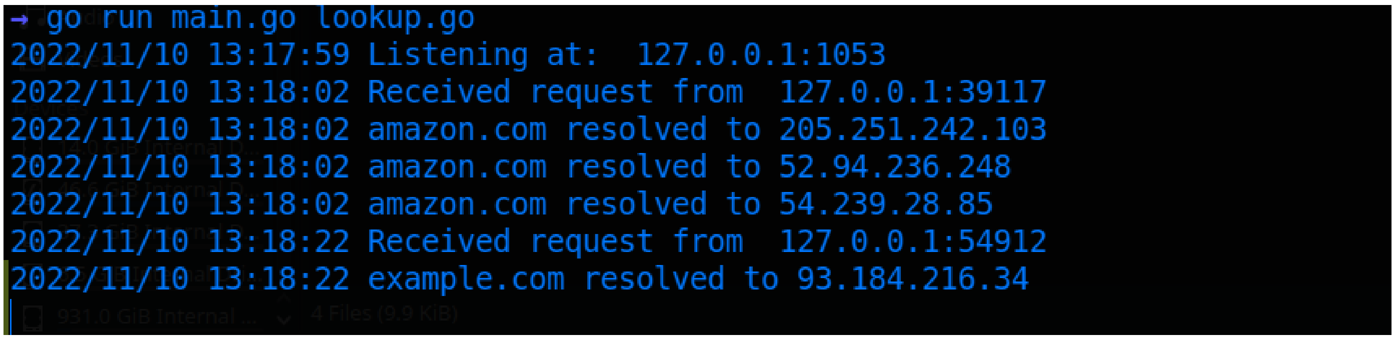
```
→ dig example.com @localhost -p 1053

; <<>> DiG 9.18.1-1ubuntu1.2-Ubuntu <<>> example.com @localhost -p 1053
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14399
;; flags: qr; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            31337   IN      A       93.184.216.34

;; Query time: 0 msec
;; SERVER: 127.0.0.1#1053(localhost) (UDP)
;; WHEN: Thu Nov 10 13:18:22 IST 2022
;; MSG SIZE  rcvd: 56
```

Fig: Resolving example.com



```
→ dig amazon.com @localhost -p 1053

; <<>> DiG 9.18.1-1ubuntu1.2-Ubuntu <<>> amazon.com @localhost -p 1053
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55261
;; flags: qr; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;amazon.com.                    IN      A

;; ANSWER SECTION:
amazon.com.             31337   IN      A       205.251.242.103
amazon.com.             31337   IN      A       52.94.236.248
amazon.com.             31337   IN      A       54.239.28.85

;; Query time: 0 msec
;; SERVER: 127.0.0.1#1053(localhost) (UDP)
;; WHEN: Thu Nov 10 13:18:02 IST 2022
;; MSG SIZE  rcvd: 106
```

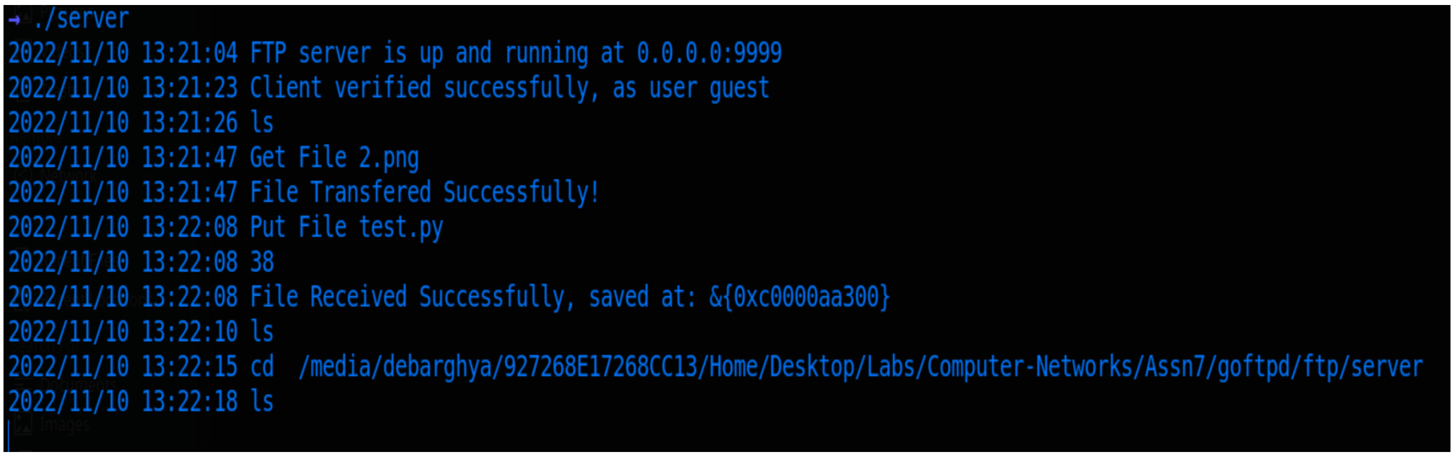
Fig: Resolving amazon.com



```
→ go run main.go lookup.go
2022/11/10 13:17:59 Listening at:  127.0.0.1:1053
2022/11/10 13:18:02 Received request from  127.0.0.1:39117
2022/11/10 13:18:02 amazon.com resolved to 205.251.242.103
2022/11/10 13:18:02 amazon.com resolved to 52.94.236.248
2022/11/10 13:18:02 amazon.com resolved to 54.239.28.85
2022/11/10 13:18:22 Received request from  127.0.0.1:54912
2022/11/10 13:18:22 example.com resolved to 93.184.216.34
```

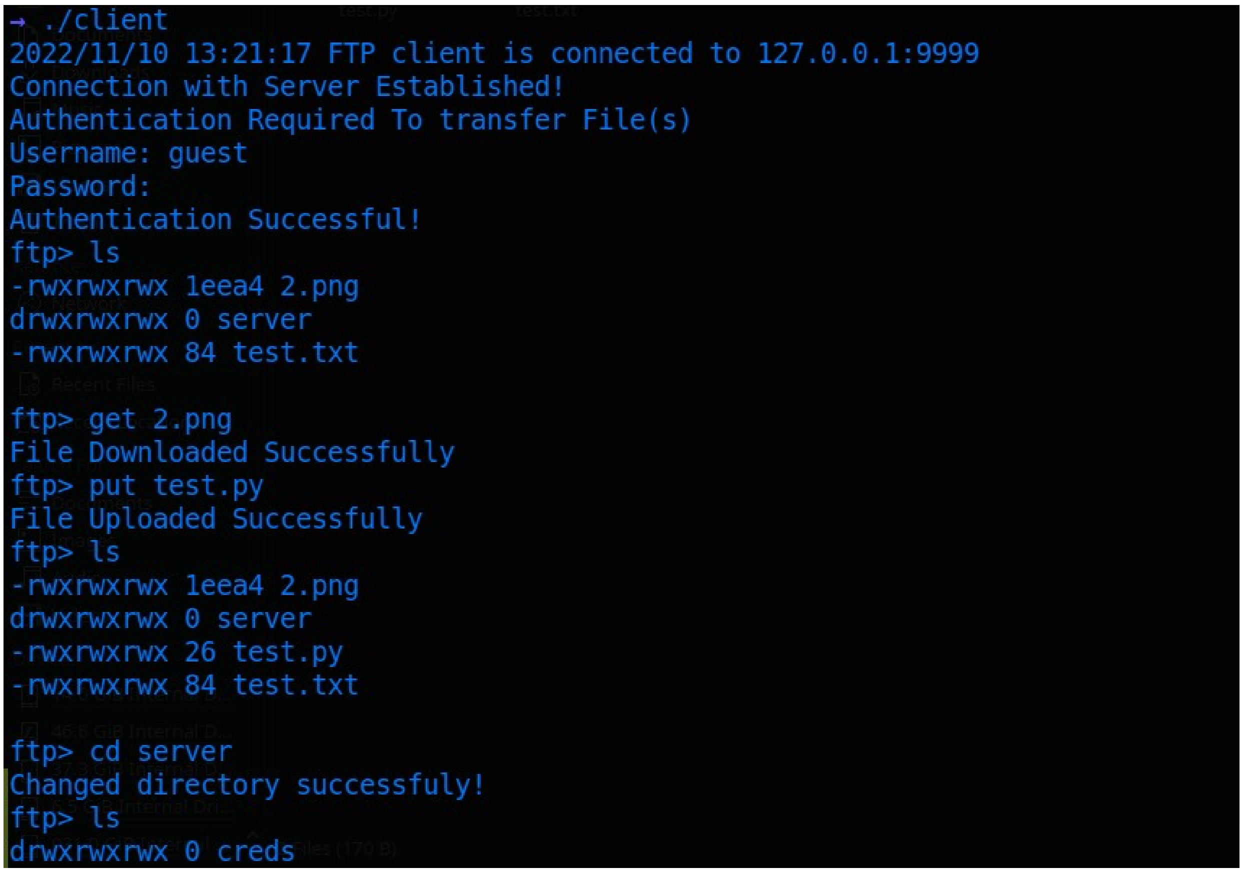
Fig: DNS Server logs

2) FTP testing: To test the FTP client and server, all the commands have been tested once separately.



```
→ ./server
2022/11/10 13:21:04 FTP server is up and running at 0.0.0.0:9999
2022/11/10 13:21:23 Client verified successfully, as user guest
2022/11/10 13:21:26 ls
2022/11/10 13:21:47 Get File 2.png
2022/11/10 13:21:47 File Transfered Successfully!
2022/11/10 13:22:08 Put File test.py
2022/11/10 13:22:08 38
2022/11/10 13:22:08 File Received Successfully, saved at: &{0xc0000aa300}
2022/11/10 13:22:10 ls
2022/11/10 13:22:15 cd  /media/debarghya/927268E17268CC13/Home/Desktop/Labs/Computer-Networks/Assn7/goftpd/ftp/server
2022/11/10 13:22:18 ls
```

Fig: FTP server logs



```
→ ./client
2022/11/10 13:21:17 FTP client is connected to 127.0.0.1:9999
Connection with Server Established!
Authentication Required To transfer File(s)
Username: guest
Password:
Authentication Successful!
ftp> ls
-rwxrwxrwx 1eea4 2.png
drwxrwxrwx 0 server
-rwxrwxrwx 84 test.txt

ftp> get 2.png
File Downloaded Successfully
ftp> put test.py
File Uploaded Successfully
ftp> ls
-rwxrwxrwx 1eea4 2.png
drwxrwxrwx 0 server
-rwxrwxrwx 26 test.py
-rwxrwxrwx 84 test.txt

ftp> cd server
Changed directory successfuly!
ftp> ls
drwxrwxrwx 0 creds
```

Fig: FTP client

ANALYSIS

a) The DNS server can resolve any domain name given that the domain name is a valid one, else it may behave unpredictably; hence scope of improvement could be adding an error handling part.
b) The DNS server is made only to handle type A requests and resolves only to IPv4 addresses, this can be improved to handle any type of requests i.e type AAAA, MX, CNAME, etc. Also the additional resource record or migration of domain names aren’t controlled too, this also can be a scope of improvement.
c) For FTP, a better authentication system than AES256-CBC can be used or the default storing of entries can be migrated to a database for faster fetching rather than parsing the JSON file again and again.
d) More commands can be added to make it more like actual FTP daemon. Also for security reasons, rather than just restricting to the ftp directory, a different FTP group must be created and carefully configured to make it more secured.
e) Migration to vSFTP or SFTP is also possible if the data sent and receved can me modified somehow.

COMMENTS

Honestly, this was the most interesting assignment of all to design and implement as this was more realistic and applicable for real world. In my opinion all other assignments should be replaced or made optional as they are no way practical or applicable to real world; also a person can’t just get the real aspect by just the simulation of stuffs.