DNS SERVER SYSTEM

CSC3011 PROJECT

(LANGUAGE OF IMPLEMENTATION: PYTHON3)

Jordan Sichalwe
Comp #: 2016134595

Contents

REQUIREMENTS

DNS App requires the following:

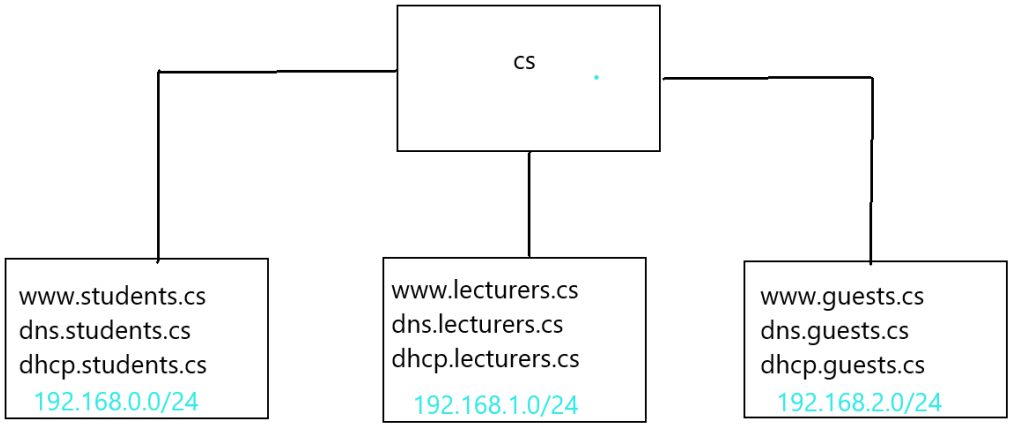
- ❖ Windows 64-bit for this build (Can be modified to work on MAC and 32-bit Windows)
- ❖ Python 3.6
    - Python Interpreter
    - Numpy module
    - Kivy modules(kivy, kivy.deps.glew, kivy.deps.sdl2)

INTRODUCTION

A ***Domain Name System*** (DNS) is a system that keeps Domain names and their corresponding IP Address in a Network. It is able to resolve domain names to their IP Addresses. This is used most due to the fact that names are more convenient for humans to remember and work with, despite this computers communicate using IP address that can easily translate to binary to allow these conversations. The DNS server resolves these names into IP Addresses before sending the packets.

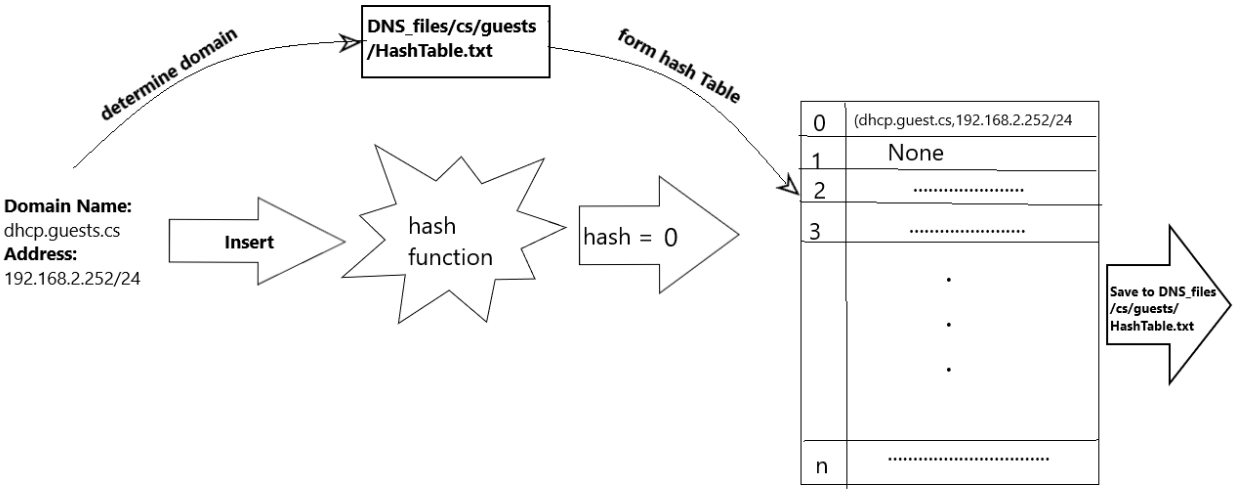
A Domain Name is a descriptive name given to a network device e.g. Computer based on a hierarchical decentralized naming system with structure <name>.<subdomain>.<upper domain> e.g. **dns.lectures.cs**

This DNS Server System uses the same approach to store domain names and their corresponding addresses to keep track of domains and their sub-domains. This also gives it the ability to resolve domain names into address. Additional Functionalities are as in a DNS Server. These include DNS Management and Creation.



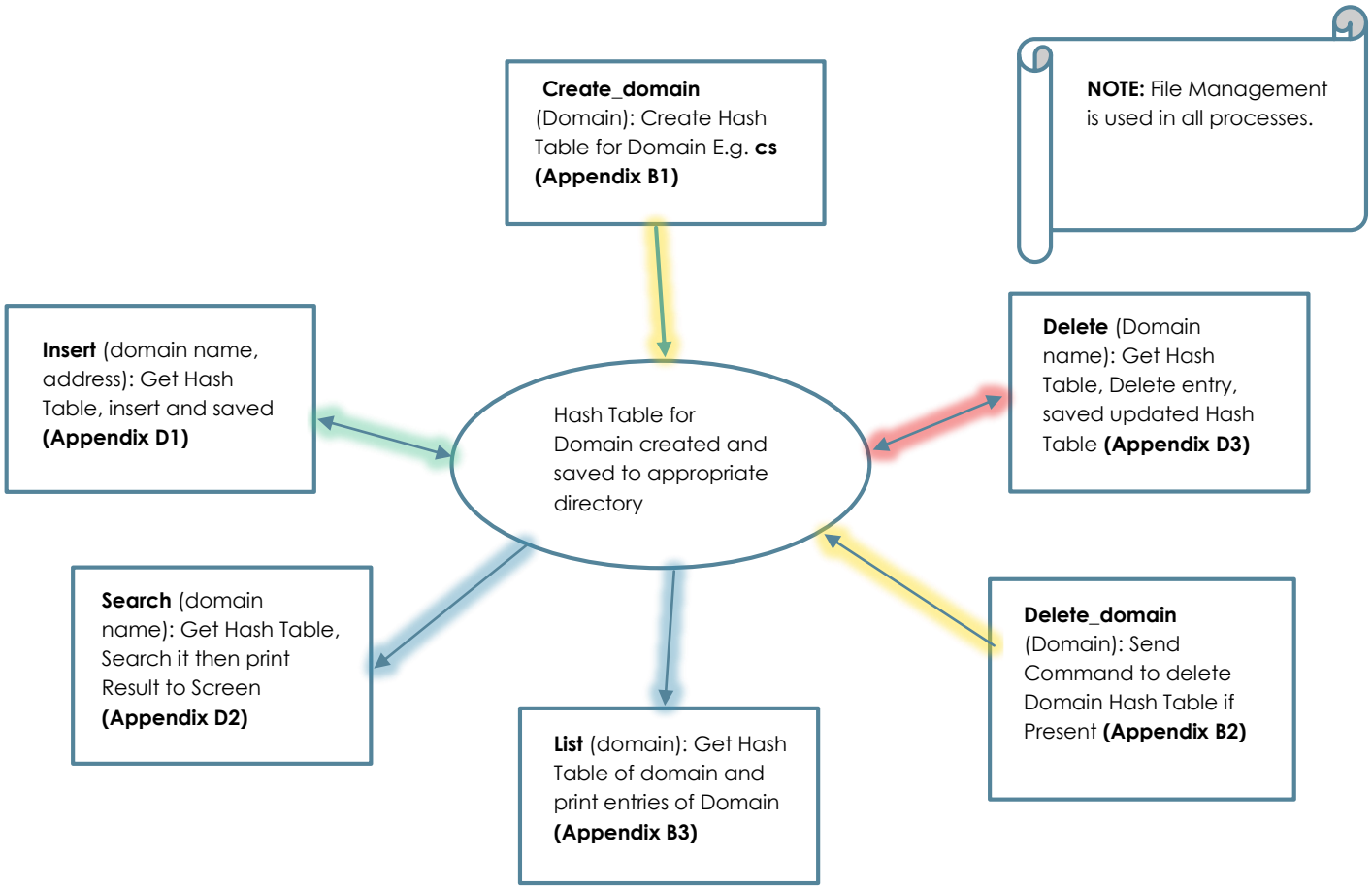
HOW IT WORKS

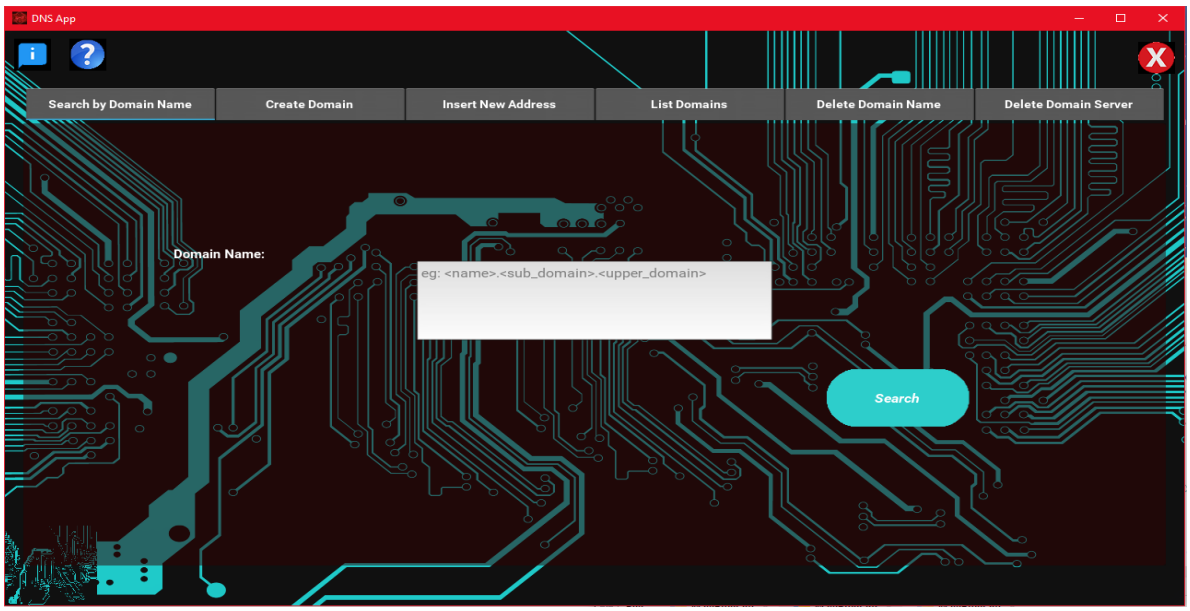
This DNS Server System uses a **Hash Table Directory System (HTDS)** to keep track of entries that are composed of domain names and their corresponding addresses. This System uses a Kivy User Interface provided by the python module Kivy. For the logic of functions, it uses an Array implementation of a Hash Table combined with an OS directory management system. The System creates directories for each new domain create and sub-domains if created. For each domain, it has a Hash Table file that represents the **Array implementation of the Hash Table**. In this implementation **Numpy** arrays are use as they execute faster than normal array to to the optimized states. The System automatically determines if a domain exists or not for all queries (insert, delete, search and list). If domain exists, it does as queried else it flags that domain is not present. The System uses a defined hash function for its queries to/from the Hash Table of a particular domain. In the event, where the resulting hash number is the same the System uses **open addressing collision resolution. Quadratic Probing** to be more precise. The Hash Table size is always **prime** and this collision resolution algorithm works better with that. In cases where the hash Table is full or an entry cannot find a position, the Table is **rehashed** to a table of prime size greater than twice the size of original table is found and all entries from previous table rehashed and new entry inserted.



Appendix D1: Insert. The figure above shows how the insertion is done.

FUNCTIONALITY BREAKDOWN





Hash Table Implementation

A) Hash Function

The Hash Function is one of the most important parts of a Hash Table. It is responsible for determining of the position on which an entry is placed in The Hash Table. It manipulates the entry to get a **Hash Number** that is used for hashing (addressing) in the table. This DNS Server System using The Ascii equivalent of the domain name to determine the hash number. To get the hash number, the hash function divides the ASCII number generated by the size of the hash table to be hashed.

hash_function: h (name) = Hash_num

$$\text{Hash\_num} = \frac{\textbf{ASCII}\ (\text{domain name})}{\text{Size of hash table}}$$

As shown that is how the hash number is obtained and then later used to determine possible position of entry in Hash Table.

i) ASCII Conversion

American Standard Code for Information Interchange (ASCII). The ASCII code uses 7 bits to represent all the alphanumeric data used in computer I/O. The same concept is used here with the help of ordinal convert in with any alphanumeric, character, number etc. can be converted to a number of equivalence and used in our hash function to get a hash number. In this Implementation all characters in domain name are converted to ordinal numbers and concatenated to form a single integer.

B) Collision Resolution using Open Addressing

A collision is a case in which entries hash to the same position. The probability of this happening is dependent on entries and the size of Hash Table. Open Addressing is one of the solution to resolve this issue and enable entries to still be placed in Hash Table. Open Addressing can be achieve through Linear Probing, Quadratic Probing and more methods. This DNS Server System uses the Quadratic Probing which is most favorable for Prime Sized Hash Tables and leaves spaces when probing. As a result, Rehashing occurs mostly when only one field is empty or table is full. Quadratic Probing works be generating a new hash number for entry that collides with already existing entities in Hash Table. For example, if hash = 0,

New_hash = 1, if collision: New_hash = 4, then 8 and so on

Until not collision occurs or there is need for rehashing.

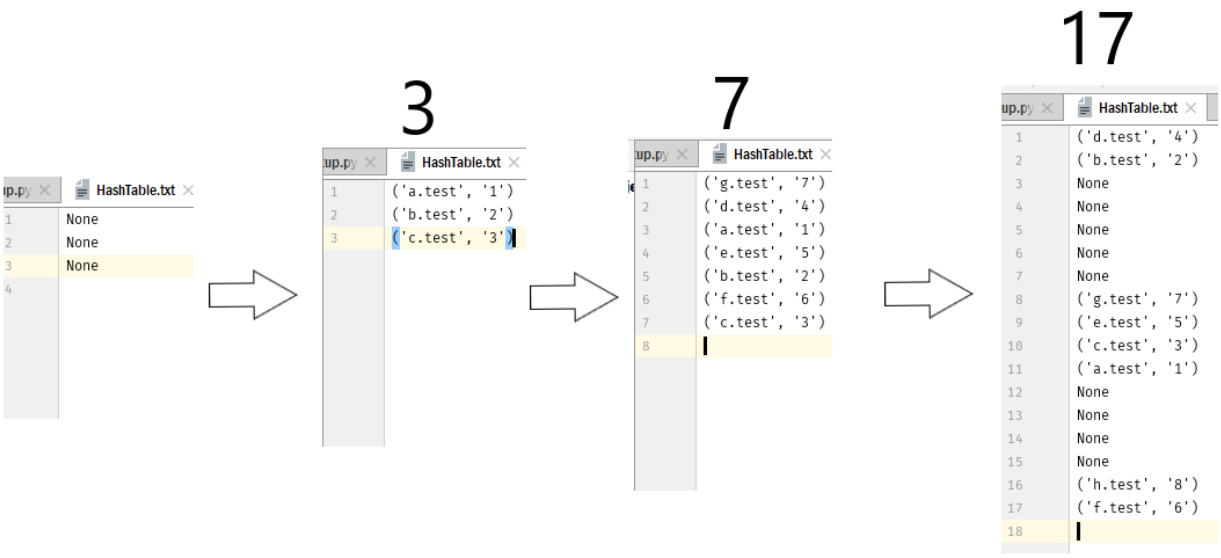
I = 1, While collision:

$$\textbf{hash\_num} = \textbf{hash\_num}_1 + I^2$$

.....check..... I++

C) Rehashing and Prime Sizing

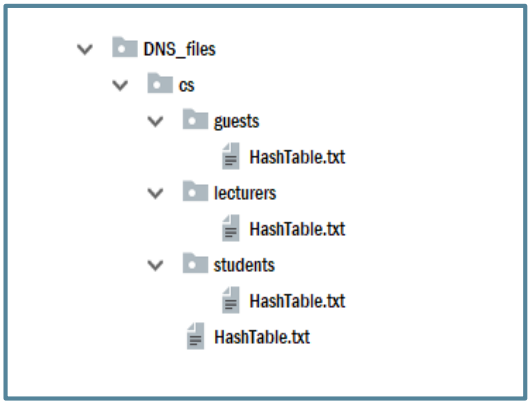
Rehashing is the situation in which the Hash Table is changed, most likely in structure. This usually occurs due to Hash Table being too small to allow new entries. Therefore, Table is increased in size and entities of old table hashed to new table. This allows more entries into the table. To guarantee that every element finds a position table are of prime size. In This implementation, when table is full or entry cannot find a place the table rehashed to another table of prime size greater than double the size of original table e.g. 3 ->7-> 17. This is called Prime Double Rehashing.



File Management

The Network File Manager has three key functions which are:

i) Choosing the DNS Hash Table
ii) Saving the Hash Table
iii) Get the Hash Table

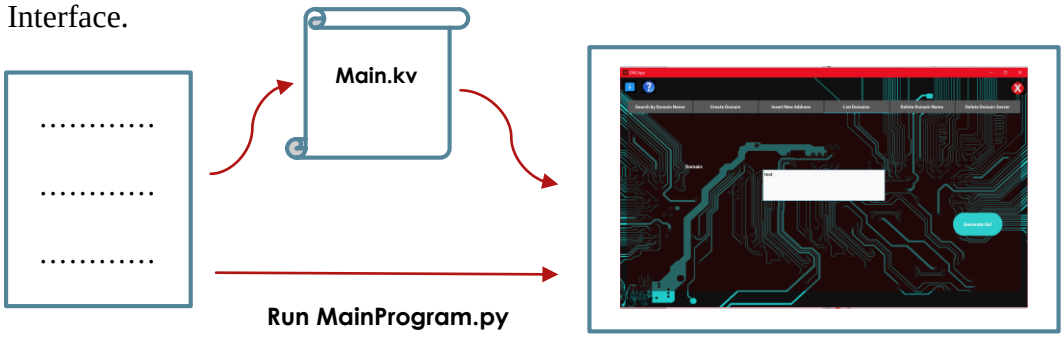


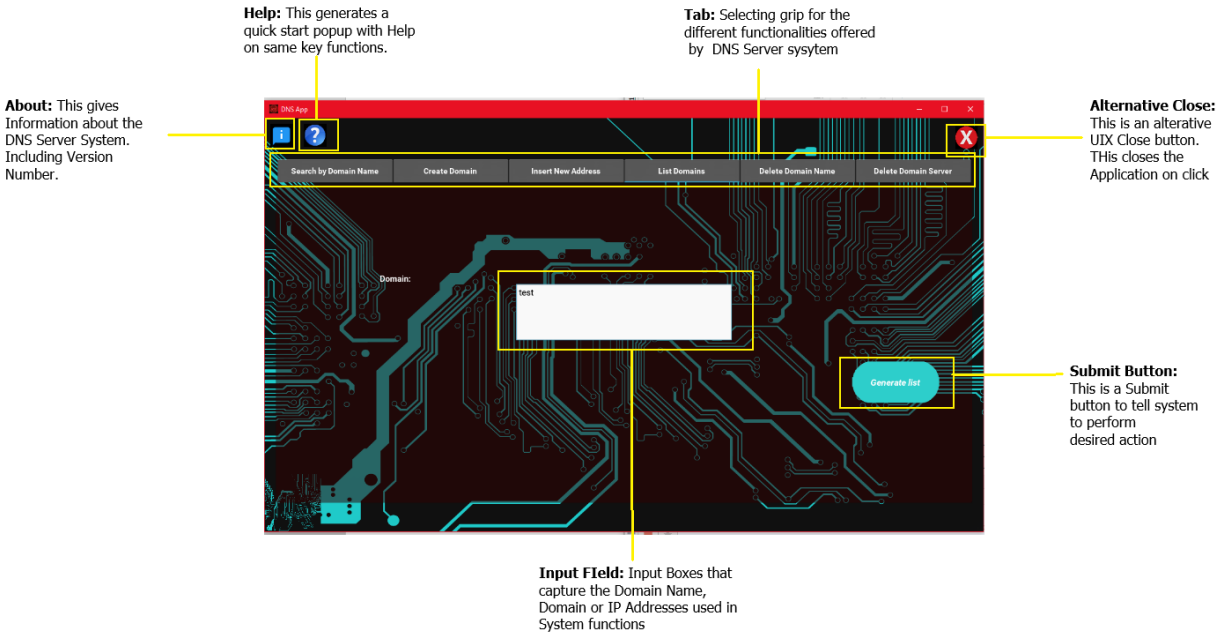
It achieves this by maintaining a directory of files as shown on the right. The HashTable.txt files Represent the Hash Tables stored on the DNS System. For **Choosing the DNS Hash Table**, it used processes the name of the input And return the directory path to the stored Hash Table (Refer to Appendix C). For example to find file for lecturers.cs It decomposes the input to an array **[‘lectures’, ’cs’]** and Returns **DNS_files/cs/lectures** at the directory. To **Save the Hash Table,** it determines the DNS directory and writes

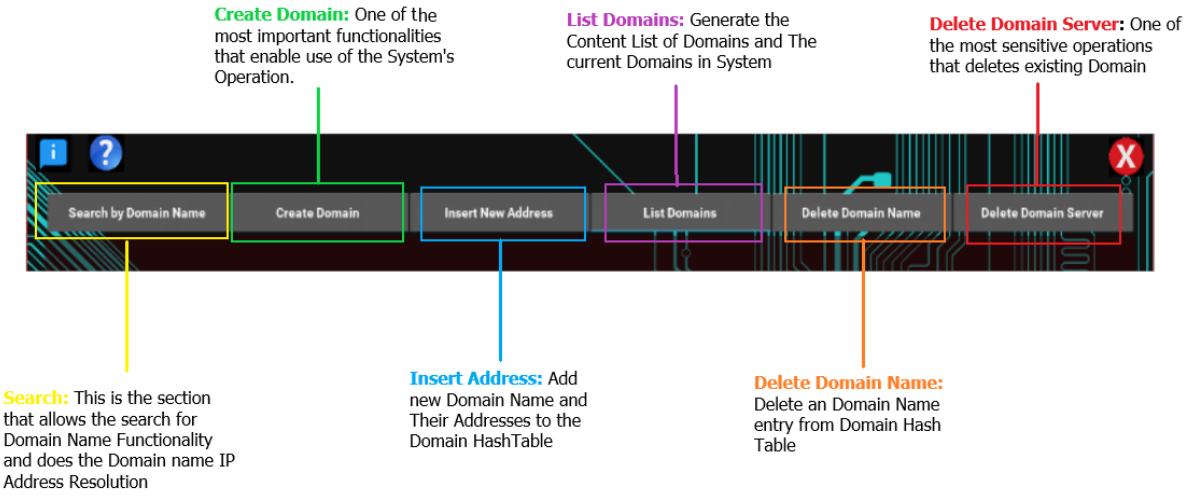
Updated Hash Table to HashTable.txt file in that directory**.**
How file is saved is as in Appendix C. **To get the Hash Table,** it firstly determine the DNS directory and if the HashTable.txt file exists, it processes it to return a Hash Table to the System for use. (Refer to Appendix C for code).

Main Program and the Kivy Language (User Interface)

The Main Program runs the entire System. It gets the DNS Server and Hash Table Objects which it uses for the Systems Logic. It uses a Grid Layout for its user interface (as in Appendix A1). The Main Program interacts with the kivy UIX and enable functionality of all Widgets on the Screen. The kivy language (Appendix A2) allows a hierarchical designing scheme to create the User Interface.







SYSTEM FUNCTIONS:

1. Create Domain

A System cannot use what it does not know or not have. Hence the requirement for create of Domain to user for our Doman Operations and to use System functionalities. A Domain name belongs to a specific Domain hence before add a Domain name its Domain or residence should be created. The System achieves this by taking in a Domain Server Input processes the name and this create a directory for the domain desired to be created. By Default, it create an empty hah table of size 17 on creation of Domain. If the Domain exists, it flags appropriately. For example, if the desired domain to be created is **google.com.** The input is processed to **[‘com’, ‘google’]** then used to create the directory **DNS_files/com/google/** .In the process it create a Hash Table storage file of size 17.

2. Insertion

i) New Entry

Adding new entries to the DNS System undergo a process similar to the one mentioned on page 4. New Entries contain a Domain Name and its corresponding IP Address. The System starts by determine the Domain of the Domain name entered and check if its Hash Table exists. If it does, it gets the Hash Table and Stores it in a hashtable array. It then generates a hash number for the new Domain, which it then uses to find a position for the new entry. The entry is stored as a tuple in the Hash Table i.e. **(Domain name, IP address).** If the hash position in the Hash Table is empty (**None**) or was deleted it inserts on that position. Otherwise, Uses Quadratic probing to determine the next possible position. Probing continues until position is

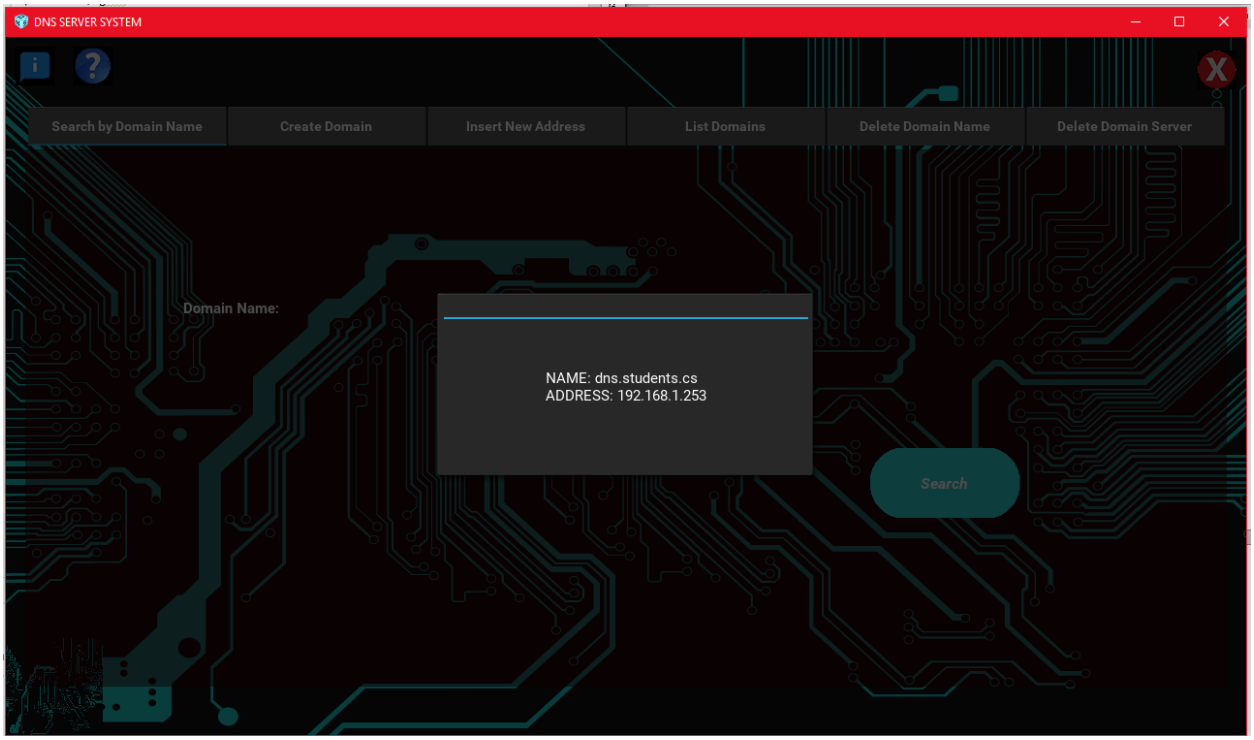
found or a conclusion that no position can be found. In the event where, position cannot be found **Rehashing** occurs as explained in section ii. Fig 2.1 shows this process.

ii) Rehashing

Rehashing as stated earlier occurs when no position can be found or if table is table is full and new entry is to be added. The System uses what is termed **Prime Double Rehashing.** Which is explained on **page 7**. Once The Table Size is increased and entities on old table rehashed the insertion is recursively retried.

3. Search

For The Search Function, The system uses the file manager to choose the domain of storage of entered domain name. If the domain exists, the HashTable of that domain is retrieved and the hash number of the entered domain name calculated. This hashes the table and checks if the domain name is at that position. If not it uses quadratic probing to check all possible position. If not found, it concludes that this domain name is not present. This uses a domain specific hashing system hence no need to check upper domain. The System uses **a LRU (Least recently used)** cache of size 64. It caches all returns hence repeated queries are faster the next time. If successful, this returns the Domain name and its corresponding address. Otherwise, flags nonexistence.



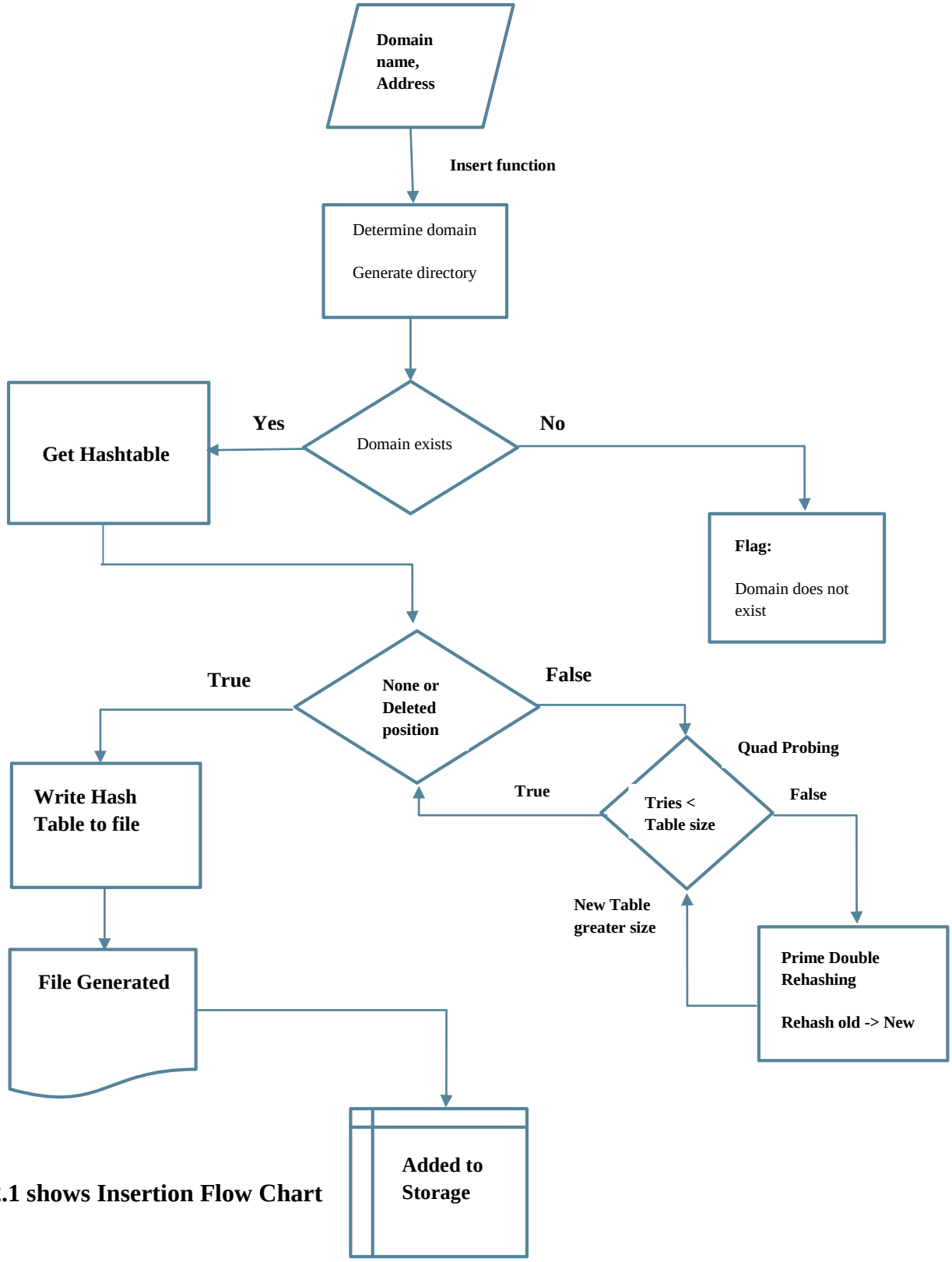
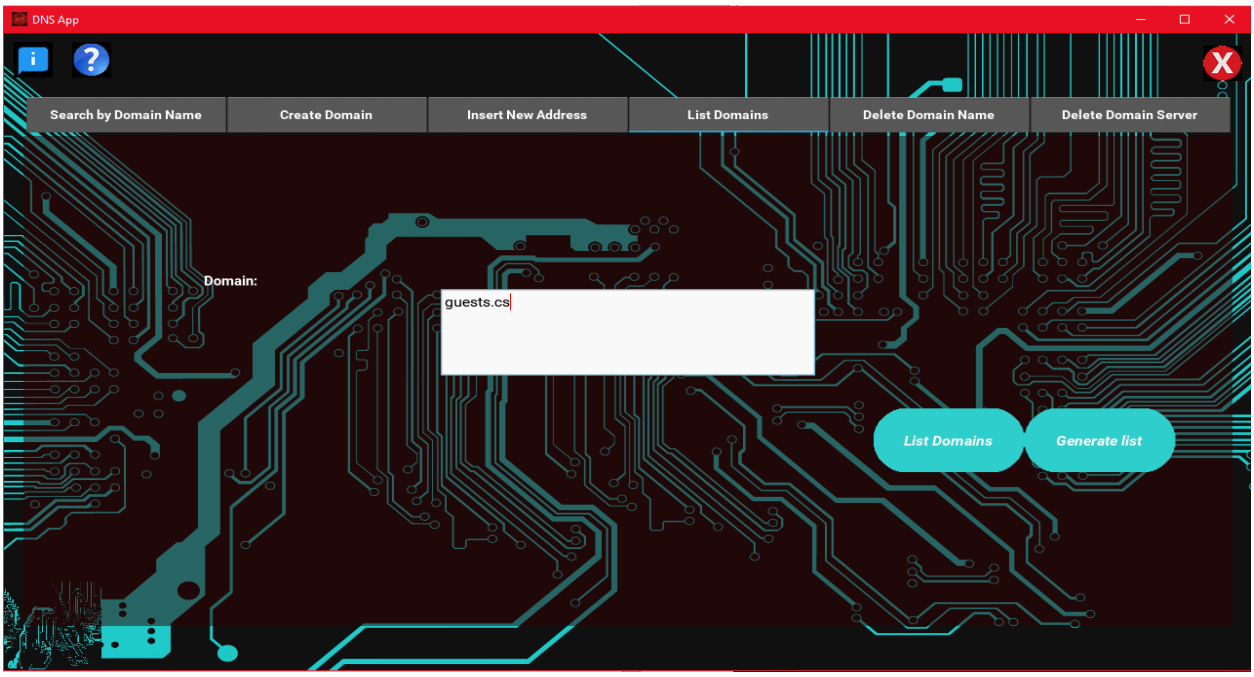
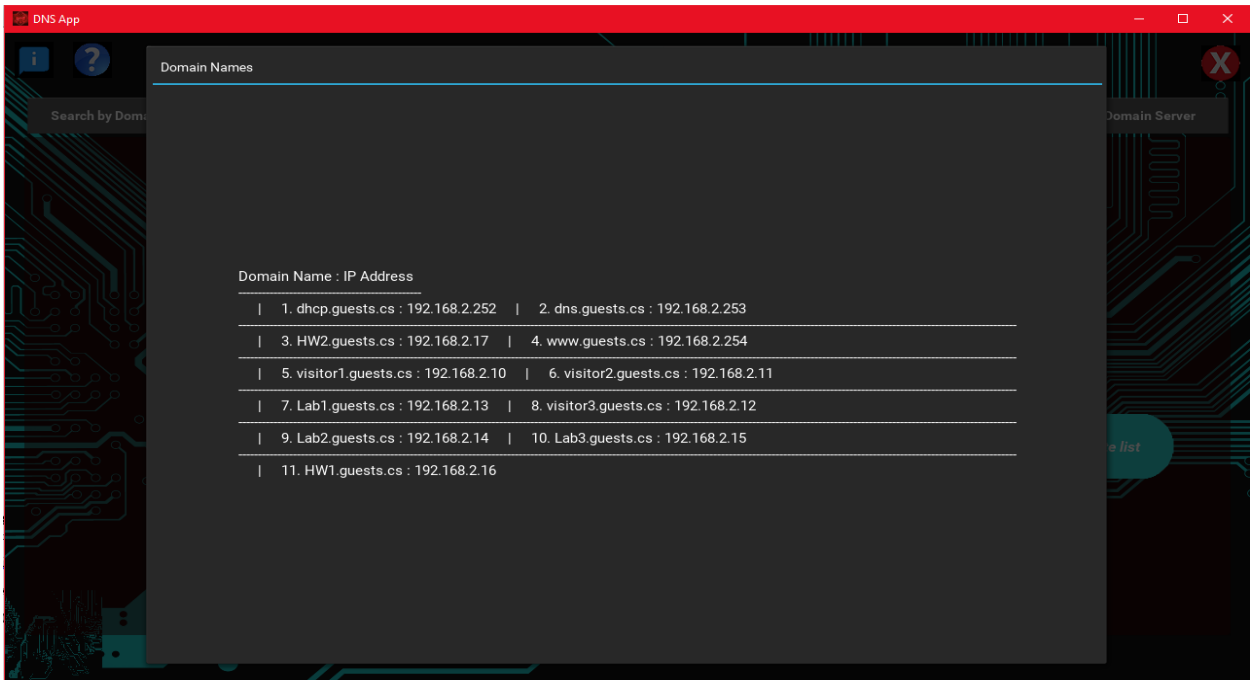


Figure 2.1 shows Insertion Flow Chart

4. List

This simply takes in the Domain from which list of entities is desired and prints them out to the screen or Allows user to generate list of Existing Domains. It does this by using file management to determine the domain Hash Table file and if it exists, processes the file and print the Domain names in the Domain with their corresponding addresses. Otherwise flags that Domain does not exist. For existing Domains, System generates all the domains in DNS.





5. Deleting Domain Name or Whole Domain

Deletion is an important functionality in every system. It removing of unwanted data or a mistake that was saved to memory. This System has a two level deletion scheme deleting of the entire Domain or Individual Domain name. These functions are achieve as below. (Refer to Appendix B2 and D3 for Code)

a) **Deletion of Individual Domain Name**
   For the deletion of an individual Domain Name, The System determines the Location of the Domain Name and generates a hash Number that is used to check possible position in Hash Table. If found, it remove the entry and Sets the Position it was in to Deleted.
b) **Deletion of Entire Domain**
c) For the deletion of an entire Domain, The System determines if the Domain directory of the file desired to be deleted is present. If present, it delete the directory using the entry as the root of the deletion. This is one of the sensitive put of the program.

CODE APPENDIXES

Appendix A: Main Program

A1: Main

```
from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.gridlayout import GridLayout
from kivy.uix.listview import ListItemButton
from kivy.uix.popup import Popup
from kivy.uix.label import Label
from kivy.uix.screenmanager import ScreenManager
from DNS_Server import Server
from NetworkHashTable import HashTable
import NetworkHashTable as netHashRef

server = Server()
newHash = HashTable()


class TabbedMenu(TabbedPanel):
    pass


class TaskBar(GridLayout):
    pass


class ListResult(ListItemButton):
    pass


class AppGridLayout(GridLayout):

    def create_domain(self, entry):
        if entry:
            try:
                netHashRef.popuptext(server.create_domain(entry))
            except (AttributeError, FileNotFoundError):
                pass

    def list_domains(self, entry):
        if entry:
            try:
                netHashRef.popuptext(server.print_directory(entry))
            except (AttributeError, FileNotFoundError, ValueError):
                if ValueError:
                    pass
                else:
                    netHashRef.popuptext("Error or No List Found")

    def generate_domains(self):
            try:
```

```
            server.print_domains(self)
        except (AttributeError, FileNotFoundError, ValueError):
            if ValueError:
                pass
            else:
                netHashRef.popuptext("Error or No List Found")

def delete_domain_server(self, entry):
    if entry:
        try:
            netHashRef.popuptext(server.delete_domain(entry))
        except (AttributeError, FileNotFoundError, ValueError):
            if ValueError:
                pass
            else:
                netHashRef.popuptext("Error,Can't Delete Domain or doesn't exist")

def insert_new(self, domain, ip_add):
    if domain and ip_add:
        try:
            netHashRef.popuptext(newHash.insert(domain, ip_add))
        except (AttributeError, FileNotFoundError, ValueError):
            if ValueError:
                pass
            else:
                netHashRef.popuptext("Error,Can't insert new entry")

def search_by_name(self, entry):
    if entry:
        try:
            newHash.search(entry)
        except (AttributeError, FileNotFoundError):
            netHashRef.popuptext("Error or Not Found")

def search_by_address(self, entry):
    if entry:
        try:
            netHashRef.popuptext(newHash.search(entry))
        except (AttributeError, FileNotFoundError):
            netHashRef.popuptext("Error or Not Found")

def delete_domain_name(self, entry):
    if entry:
        try:
            netHashRef.popuptext(newHash.delete_address(entry))
        except (AttributeError, FileNotFoundError, ValueError):
            if Exception is ValueError:
                netHashRef.popuptext("No Value Captured")
            else:
                pass

def close_window(self):
    Main.stop(App)

def list_of_domains(self):
    pass
```

```
    def delete_entire_dns_domain(self):
        pass

    def about_popup(self):
        str_text = ''
        file = open("about.txt", 'r')
        about_text = file.readlines()
        for text in about_text:
            str_text += text
        popup = Popup(title='About',
                      content=Label(
                          text=str_text),
                      size_hint=(None, None), size=(700, 500))
        popup.open()

    def help_popup(self):
        str_text = ''
        file = open("help.txt", 'r')
        about_text = file.readlines()
        for text in about_text:
            str_text += text
        popup = Popup(title='HELP',
                      content=Label(text=str_text),
                      size_hint=(None, None), size=(800, 600))
        popup.open()
        file.close()


class AppBoxLayout(BoxLayout):
    pass


class RootScreen(ScreenManager):
    pass


class MainApp(App):
    App.title = "DNS App"
    App.icon = "UI_files/icon.jpg"

    def build(self):
        return AppGridLayout()


Main = MainApp()

if __name__ == "__main__":
    Main.run()
```

A2: Kivy Design

```
#: import ListAdatpter kivy.adapters.listadapter.ListAdapter
#: import ListItemButton kivy.uix.listview.ListItemButton

AppGridLayout:

<CustButton@Button>:
    font_size: 16
    bold:True
    size: 100, 30
    background_normal: 'UI_files/button_norm.png'
    background_down: 'UI_files/button_down.png'
    background_color: .88, .88, .88, 1
    size_hint: .2, .2

<AppGridLayout>:

    id:app_grid
    rows: 5
    padding:10
    spacing:10
    canvas:
        Rectangle:
            source: 'UI_files/famous.gif'
            size: self.size
            pos: self.pos
    BoxLayout:
        padding: 0
        size_hint_y: None
        height: 40
        spacing: 20
        # Make the background for the toolbar white
        canvas:
            Rectangle:
                source: 'UI_files/famous.gif'

        Button:
            background_normal: 'UI_files/about.png'
            background_down: 'disk_dn.png'
            size_hint_x: None
            on_press:app_grid.about_popup()
            width: 40
        Button:
            background_normal: 'UI_files/help_icon.png'
            background_down: 'exit_dn.png'
            size_hint_x: None
            on_press:app_grid.help_popup()
            width: 40
        FloatLayout:
            padding:10
            Button:
                background_normal: 'UI_files/close.png'
                background_down: 'exit_dn.png'
                pos_hint: {"right":1, "y": -.1}
                size_hint_x:None
```

```
            size_hint_y:None
            height:40
            width:40
            on_press: app_grid.close_window()


    TabbedMenu:
        do_default_tab:False
        size_hint: 20, 10
        padding:10
        background_color: 1, 0, 0, .5
        tab_width:206


        TabbedPanelItem:
            font_size:14
            bold:True
            text: "Search by Domain Name"
            BoxLayout:
                size_hint_y: None
                height: "400dp"
                spacing:20
                padding:20
                Label:
                    pos_hint:{"top":.1,"y":1}
                    text: "Domain Name:"
                    bold: True
                TextInput:
                    pos_hint:{"centre_x":.5,"y":.3}
                    size_hint_y: None
                    height: "100dp"
                    hint_text:"eg: <name>.<sub_domain>.<upper_domain>"
                    id:search_domain_name
                    font_size:16
                    italic: True
                    multiline: False

                FloatLayout:
                    CustButton:
                        text: "Search"
                        pos_hint: {"right":.5, "y": 0}
                        size_hint: .4, .2
                        italic: True
                        on_press: app_grid.search_by_name(search_domain_name.text)


        TabbedPanelItem:
            font_size:14
            bold:True
            text: "Create Domain"
            BoxLayout:
                size_hint_y: None
                height: "400dp"
                spacing:20
                padding:20
                Label:
                    pos_hint:{"top":.1,"y":1}
```

```
            text: "Domain Name:"
            bold: True
        TextInput:
            pos_hint:{"centre_x":.5,"y":.3}
            size_hint_y: None
            hint_text:"eg: <sub_domain>.<upper_domain>"
            height: "100dp"
            id:create_domain_name
            font_size:16
            italic: True
            multiline: False

        FloatLayout:
            CustButton:
                text: "Create"
                pos_hint: {"right":.9, "y":0}
                size_hint: .4, .2
                italic: True
                on_press: app_grid.create_domain(create_domain_name.text)

TabbedPanelItem:
    font_size:14
    bold:True
    text: "Insert New Address"
    BoxLayout:
        size_hint_y: None
        height: "400dp"
        spacing:20
        padding:20
        Label:
            pos_hint:{"top":.1,"y":1}
            text: "Domain Name:"
            bold: True
        TextInput:
            pos_hint:{"centre_x":.5,"y":.3}
            size_hint_y: None
            height: "100dp"
            hint_text:"eg: <name>.\n<sub_domain>.\n<upper_domain>"
            id:insert_domain_name
            font_size:16
            italic: True
            multiline: False
        Label:
            pos_hint:{"top":.1,"y":1}
            text: "IP Address:"
            bold: True
        TextInput:
            pos_hint:{"centre_x":.5,"y":.3}
            size_hint_y: None
            hint_text:"eg: 100.100.100.100"
            height: "100dp"
            id:insert_ip_add
            font_size:16
            italic: True
            multiline: False
        FloatLayout:
            CustButton:
```

```
            text: "Add Address"
            pos_hint: {"right":.9, "y": 0}
            size_hint: .5, .2
            italic: True
            on_press: app_grid.insert_new(insert_domain_name.text, insert_ip_add.text)

TabbedPanelItem:
    font_size:14
    bold:True
    text: "List Domains"
    BoxLayout:
        size_hint_y: None
        height: "400dp"
        spacing:20
        padding:20
        Label:
            pos_hint:{"top":.1,"y":1}
            text: "Domain:"
            bold: True
        TextInput:
            pos_hint:{"centre_x":.5,"y":.3}
            size_hint_y: None
            height: "100dp"
            hint_text:"eg. <domain_Server>"
            id:list_domain_name
            font_size:16
            italic: True
            multiline: False

        FloatLayout:
            CustButton:
                text: "Generate list"
                pos_hint: {"right":.9, "y": 0}
                size_hint: .4, .2
                italic: True
                on_press: app_grid.list_domains(list_domain_name.text)
            CustButton:
                text: "List Domains"
                pos_hint: {"x":.1, "y": 0}
                size_hint: .4, .2
                italic: True
                on_press: app_grid.generate_domains()



TabbedPanelItem:
    font_size:14
    bold:True
    text: "Delete Domain Name"
    BoxLayout:
        size_hint_y: None
        height: "400dp"
        spacing:20
        padding:20
        Label:
            pos_hint:{"top":.1,"y":1}
```

```
            text: "Domain Name:"
            bold: True
        TextInput:
            id:delete_name
            pos_hint:{"centre_x":.5,"y":.3}
            size_hint_y: None
            height: "100dp"
            hint_text:"eg: <name>.<sub_domain>.<upper_domain>"
            font_size:16
            italic: True
            multiline: False

        FloatLayout:
            CustButton:
                text: "Delete"
                pos_hint: {"right":.9, "y": 0}
                size_hint: .4, .2
                italic: True
                on_press: app_grid.delete_domain_name(delete_name.text)


TabbedPanelItem:
    font_size:14
    bold:True
    text: "Delete Domain Server"
    BoxLayout:
        size_hint_y: None
        height: "400dp"
        spacing:20
        padding:20
        Label:
            pos_hint:{"top":.1,"y":1}
            text: "Domain Name:"
            bold: True
        TextInput:
            id:delete_input
            pos_hint:{"centre_x":.5,"y":.3}
            size_hint_y: None
            height: "100dp"
            hint_text:"eg: <sub_domain>.<upper_domain>"
            font_size:16
            italic: True
            multiline: False

        FloatLayout:
            CustButton:
                text: "Delete"
                pos_hint: {"right":.9, "y": 0}
                size_hint: .4, .2
                italic: True
                on_press: app_grid.delete_domain_server(delete_input.text)
```

Appendix B: DNS Logic

Server Object

```
import NetworkFileManager as netMang
from kivy.uix.popup import Popup
from kivy.uix.label import Label
import NetworkHashTable as netHash
import numpy as np
import os
import shutil
import re


class Server(object):
    @staticmethod
    def create_domain(domain):
        ------------------------code-----------------------------

    @staticmethod
    def delete_domain(domain_name):
        -------------------------code-----------------------------

    @staticmethod
    def print_directory(domain):
        -------------------------code-----------------------------
```

B1.create domain

```
def create_domain(domain):
    filename = domain
    filename_list = re.sub('[^\w]', " ", filename).split()
    dns_file = "./DNS_files"
    for i in range(len(filename_list) - 1, -1, -1):
        word = "/" + filename_list[i]
        dns_file += word
    check_dir = dns_file
    dns_file = dns_file + "/HashTable.txt"
    try:
        if not os.path.exists(dns_file):
            directory = os.path.dirname(dns_file)
            os.makedirs(directory)
    except FileExistsError:
        pass
    dirlist = os.listdir(check_dir)
    if filename is ".txt":
        return "Please Enter Domain Name"
    elif "HashTable.txt" in dirlist:
        return "DNS file already exists"
    else:
        file = open(dns_file, 'w')
        default_length = 17
        arr = np.array([None] * default_length)
        for x in range(arr.size):
            file.write("{}\n".format(arr[x]))
```

```
        file.close()
        return "DNS_Server Created"
```

B2. Delete domain

```
def delete_domain(domain_name):
    filename_list = re.sub('[^\w]', " ", domain_name).split()
    dns_file = "./DNS_files"
    for i in range(len(filename_list) - 1, -1, -1):
        word = "/" + filename_list[i]
        dns_file += word
    try:
        directory = dns_file
        dirlist = os.listdir(dns_file)
        dns_file += "/HashTable.txt"
    except FileNotFoundError:
        return "DNS not Found"
    if "HashTable.txt" in dirlist:
        shutil.rmtree(directory)
        netHash.popuptext("Domain Deleted")
    else:
        return "Not Found"
```

B3. List entries in domain

```
def print_directory (domain):
    domain = "dammy." + domain
    text = 'Domain Name : IP Address \n----------------------------------------------\n'
    hashtable = netMang.get_file(netMang.choose_dns(domain))
    count = 0
    for entry in range(len(hashtable)):
        if hashtable[entry] is None:
            continue
        elif hashtable[entry].__contains__('Deleted'):
            continue
        else:
            gen_tuple = eval(hashtable[entry])
            text += "    |    " + str(count + 1) + ". " + gen_tuple[0] + " : " + gen_tuple[1]
            count += 1
            if count % 2 == 0:
                text += "\n-----------------------------------------------------------" \
                        "-----------------------------------------------------------" \
                        "-----------------------------------------------------------\n"
    popup = Popup(title='About',
                  content=Label(text=text),
                  size_hint=(None, None), size=(1000, 700))
    popup.open()
```

B4. Print Domains

```
def print_domains(self):
    root_dir = 'DNS_files'
    count = 0
    str_dom = ''
    text = ''
    for dirName, subdirList, fileList in os.walk(root_dir):
        if dirName == 'DNS_files':
            continue
        filename_list = re.sub('[^\w]', " ", dirName).split()
        for i in range(len(filename_list) - 1, 0, -1):
            str_dom += filename_list[i] + "."
        text += '  {}. {} |'.format(count + 1, str_dom)
        count += 1
        str_dom = ''
        if count % 4 == 0:
            str_dom += "\n"
    popup = Popup(title='Domains',
                  content=Label(text=text),
                  size_hint=(None, None), size=(1000, 700))
    popup.open()
```

Appendix C: File Management

```
import os
import numpy as np
import re


def write_file(domain, array):
    dns_server = choose_dns(domain)
    filename = dns_server + "/HashTable.txt"
    file = open(filename, 'w')
    listtofile = array
    for item in listtofile:
        file.write("{}\n".format(item))
    file.close()


def get_file(server):
    server = server + "/HashTable.txt"
    arr = np.array([])
    empty = np.array([None])
    try:
        if os.path.exists(server):
            name = server
            file = open(name, 'r')
            content = file.readlines()
            for x in content:
                entry = x.strip('\n')
                if entry == 'None':
                    entry = None
                arr = np.append(arr, entry)
```

```
            return arr
        else:
            return empty
    except FileNotFoundError:
        return "Fetch Error"


def choose_dns(name):
    char_arr = []
    count = 0
    ch = ''
    if len(name) < 1:
        return "Invalid name"
    else:
        filename_list = re.sub('[^\w]', " ", name).split()
        dns_file = "./DNS_files"
        for i in range(len(filename_list) - 1, 0, -1):
            word = "/" + filename_list[i]
            dns_file += word
        ch = dns_file
    return ch
```

Appendix D: Hash Table

Hash Table Object

```
import numpy as np
import NetworkFileManager as netMang
from kivy.uix.popup import Popup
from kivy.uix.label import Label
from functools import lru_cache

# uses open addressing collison resolution
hashcode = 0
'''
Uses System Cache Least Recently Used = lru_cache
'''


@lru_cache(maxsize=32)

class HashTable(object):
    array = np.array([None])
    def __init__(self):
        self.array = np.array([None] * 17)

    def in_cache(self, element):
        ----------code----------

    def insert(self, domain_name, ip_address):
        ----------code----------

    def search(self, element):
        ----------code----------
```

```
    def delete_address(self, element):
        ------------------code-------------------------

    ------------------------------cont-----------------------------------------
```

D1: Insert

```
def insert(self, domain_name, ip_address):
    hashtable = netMang.get_file(netMang.choose_dns(domain_name))
    hash_number = hash_func(domain_name, hashtable)
    if hashtable.size < 2:
        return "Domain does not exists "
    # inserted = False
    if hashtable[hash_number] is None:
        hashtable[hash_number] = (domain_name, ip_address)
        netMang.write_file(domain_name, hashtable)
        return "{} insertion successful".format(domain_name)
    elif hashtable[hash_number].__contains__('Deleted'):
        hashtable[hash_number] = (domain_name, ip_address)
        netMang.write_file(domain_name, hashtable)
        return "{} insertion successful".format(domain_name)
    else:
        check = eval(hashtable[hash_number])
        check = str(''.join(check[0]))
        if domain_name != check:
            i = 1
            while hashtable[hash_number] is not None and not hashtable[hash_number].__contains__('Deleted'):
                hash_number = quadratic_probing(i, domain_name, hashtable)
                i += 1
                if i == hashtable.size:
                    hashtable = self.rehash_table(domain_name, hashtable)
                    self.insert(domain_name, ip_address)
                    break

            else:
                hashtable[hash_number] = (domain_name, ip_address)
                netMang.write_file(domain_name, hashtable)
                return "{} insertion successful".format(domain_name)

        else
            return domain_name + " already exists"
    self.array = hashtable
```

D2: Search

```
def search(self, element):
    search_hashtable = netMang.get_file(netMang.choose_dns(element))
    hash_number = hash_func(element, search_hashtable)
    if search_hashtable[hash_number] is None:
        popuptext("{} is not on DNS Server".format(element))
    else:
```

```
    while search_hashtable[hash_number] is not None:
        tuple = eval(search_hashtable[hash_number])
        check = str(''.join(tuple[0]))
        if check == element:
            popuptext("NAME: {}\nADDRESS: {}\n".format(tuple[0], tuple[1]))
            break
        else:
            i = 1
            check = str(''.join(tuple[0]))
            while check != element:
                hash_number = quadratic_probing(i, element, search_hashtable)
                i += 1
                if search_hashtable[hash_number] is None:
                    pass
                    break
                tuple = eval(search_hashtable[hash_number])
                check = str(''.join(tuple[0]))
                if i == search_hashtable.size:
                    break
            else:
                pass

    else:
        popuptext("{} is not on DNS Server".format(element))
```

D3: delete

```
def delete_address(self, element):
    delete_from_hashtable = netMang.get_file(netMang.choose_dns(element))
    hash_number = hash_func(element, delete_from_hashtable)
    if delete_from_hashtable[hash_number] is None:
        popuptext("{} is not on DNS Server".format(element))
    else:
        while delete_from_hashtable[hash_number] is not None:
            tuple = eval(delete_from_hashtable[hash_number])
            check = str(''.join(tuple[0]))
            if check == element:
                delete_from_hashtable[hash_number] = ("Deleted", "None")
                netMang.write_file(element, delete_from_hashtable)
                popuptext("{} Deleted from DNS".format(check))
                break
            else:
                i = 1
                check = str(''.join(tuple[0]))
                while check != element:
                    hash_number = quadratic_probing(i, element, delete_from_hashtable)
                    i += 1
                    if delete_from_hashtable[hash_number] is None:
                        pass
                        break
                    tuple = eval(delete_from_hashtable[hash_number])
                    check = str(''.join(tuple[0]))
                    if i == delete_from_hashtable.size:
                        break
```

```
            else:
                delete_from_hashtable[hash_number] = ("Deleted", "None")
                netMang.write_file(element, delete_from_hashtable)
                popuptext("{} Deleted from DNS".format(check))

        else:
            popuptext("{} is not on DNS Server or Was Deleted".format(element))
```

D4: Rehash

```
def rehash_table(self, domain_name, hashtable):
    temp = hashtable
    size_of_hashtable = next_prime(temp.size * 2)
    domain = netMang.choose_dns(domain_name)
    arr = np.array([None] * size_of_hashtable)
    netMang.write_file(domain_name, arr)
    hashtable = netMang.get_file(netMang.choose_dns(domain_name))
    for i in range(temp.size):
        if temp[i] is None:
            continue
        else:
            element = eval(temp[i])
            self.insert(element[0], element[1])

    return hashtable
```

D5: Quadratic Probing

```
def quadratic_probing(iteration, element, hashtable):
    hash_number = hash_func(element, hashtable)
    hash_number = (hash_number + iteration ** 2) % hashtable.size
    return hash_number
```

D6: Hash Function

```
def hash_func(element, array):
    element = ascii_converter(element)
    hash_number = element % array.size
    return hash_number
```

D7: Next Prime Algorithm

```
def next_prime(num):
    i = 2
    while i < num:
        if num % i == 0:
            num += 1
            i = 2
        elif i == num // 2:
            break
        else:
            i += 1
            print(i)

    return num
```

D8: Ascii Converter

```
def ascii_converter(entry):
    entry_result = int(''.join(str(ord(char)) for char in entry))
    return entry_result
```

D9: Popup Text Widget

```
def popuptext(text_input):
    popup = Popup(title='',
                  content=Label(text=text_input),
                  size_hint=(None, None), size=(400, 200))
    popup.open()
```