DNS实验报告

姓名：郑雨霏

学号：PB19050946

实验环境

实验设备: MacBook Pro

实验环境: macOS 11.4 Visual Studio Code 1.16.2

解释程序

先利用套接字获得消息，利用handle函数处理报文，handle函数中调用DNS_Packege函数处理数据，判断报文类型，如果在example.txt中，则利用generate_response函数获得相应响应报文，转发回去，否则转发出去，再接收响应报文转发回去

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import socket
import threading
from time import time


class DNS_Relay_Server:      #一个relay server实例，通过缓存文件和外部地址来初始化
    def __init__(self,cache_file,name_server):
        #url_IP字典:通过域名查询ID
        self.url_ip = {}
        self.cache_file = cache_file
        self.load_file()
        self.name_server = name_server
        #trans字典：通过DNS响应的ID来获得原始的DNS数据包发送方
        self.trans = {}

        #加载文件
    def load_file(self,):
        f = open(self.cache_file,'r',encoding='utf-8')#通过读的方式打开文件
        for line in f:
            ip,name = line.split(' ')#利用空格将ip和名字隔开
            self.url_ip[name.strip('\n')] = ip#用换行符将name隔开，并与ip对应放入字典
中
```

```
        f.close()#关闭文件

        #运行dns
    def run(self):
        buffer_size = 512#将读取的大小设置为512
        #socke配置和端口关联
        server_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        server_socket.bind(('',53))
        server_socket.setblocking(False)
        #不断读取信息并利用handle函数进行处理
        while True:
            try:
                data,addr = server_socket.recvfrom(buffer_size)
                threading.Thread(target=self.handle,args=
(server_socket,data,addr)).start()
            except:
                continue

        #处理dns报文
    def handle(self,server_socket,data,addr):
      #调用DNS_Package类来解析data里面的数据，如报文id、类型、query中的名字等
        RecvDp = DNS_Packege(data)
        id = RecvDp.ID
        name = RecvDp.name;
        #如果是请求报文，判断名字是否在example.txt文件中，即字典url_ip中，且是一个请求报文，
若存在则本地生产应答报文，否则将消息转发出去
        print(name)#输出获得的报文中的名字，用于debug
        if not RecvDp.QR :
            if name in self.url_ip and RecvDp.type == 1:
              #如果是个设置为无效的ip，则将Intercepted标记为1，否则为0
                if self.url_ip[name] == '0.0.0.0':
                    Intercepted = 1
                else :
                    Intercepted = 0
               #调用generate_response函数生成应答报文，将name对应的ip传入函数中
                respond =
RecvDp.generate_response(self.url_ip[name],Intercepted)
               #发送回请求报文中给定的域名服务器和端口
                server_socket.sendto(respond,addr)
            else:
              #否则将报文转发出去，并记录下请求报文中的域名服务器和查询的名字
                server_socket.sendto(data, self.name_server)
                self.trans[id] = (addr, name)
            #statement
        #如果不是请求报文，则转发回原来存在trans中的地址，转发完成后删除
        if RecvDp.QR :
            if id in self.trans:
                target_addr, name = self.trans[id]
                server_socket.sendto(data, target_addr)
                del self.trans[id]
            #statement

```

```python
#解析报文
class DNS_Packege:        #一个DNS Frame实例，用于解析和生成DNS帧
    def __init__(self,data):
        Msg_arr = bytearray(data)
        #ID
        self.ID = (Msg_arr[0] << 8 ) +Msg_arr[1]
        # FLAGS
        self.QR = Msg_arr[2] >> 7
        # 资源记录数量
        self.QDCOUNT = (Msg_arr[4] << 8) + Msg_arr[5]
        #self.ANSWER = (Msg_arr[6] << 8) + Msg_arr[7]
        self.AUTHOR = (Msg_arr[8] << 8 ) + Msg_arr[9]
        self.ADDI = (Msg_arr[10] << 8 ) + Msg_arr[11]
        #query内容解析
        """data -> name, querybytes, type, classify, len"""
        self.name = []
        self.name_length = 0
        name_block = int(Msg_arr[12])
        part = ''
        #通过实验文档中所给的格式（长度+字符+长度+字符+……+\x0)解析name
        while(name_block != 0):
            i = 1;
            if( self.name_length != 0 ):
                self.name.append(".")
            while i <= name_block:
                #part = part + (chr(data[12 + self.name_length + i] ))
                self.name.append(chr(data[12 + self.name_length + i] ))
                i = i + 1
            #self.name.append(part)
            #part = ''
            self.name_length = self.name_length + name_block + 1
            name_block = int(Msg_arr[12 + self.name_length])
       #将list类型转化为str类型，并解析剩下的信息
        self.name = ''.join(self.name)
        self.name_length = self.name_length + name_block + 1;
        self.type = (Msg_arr[12 + self.name_length ] <<8) + Msg_arr[12 +
self.name_length + 1]
        self.classify = (Msg_arr[12 + self.name_length + 2] <<8) + Msg_arr[12 +
self.name_length + 3];
        self.len = self.name_length + 4
        self.data = data



    #生成回答
    def generate_response(self,ip,Intercepted):
      #如果不是无效的ip，则按照格式生成响应报文
        if not Intercepted:
            #初始化res，并设定长度
            res = bytearray(32 + self.name_length)
            #ID
            res[0] = self.ID >> 8
```

```
            res[1] = self.ID % 256
            #FLAGS
            res[2] = 0x81
            res[3] = 0x80
            # 资源记录数量
            res[4] = self.QDCOUNT >> 8
            res[5] = self.QDCOUNT % 256
            res[6] = 0x00
            res[7] = 0x01
            res[8] = self.AUTHOR >> 8
            res[9] = self.AUTHOR % 256
            res[10] = self.ADDI >> 8
            res[11] = self.ADDI % 256
            #query内容解析
            for i in range(12, 16 + self.name_length):
                res[i] = self.data[i]
            #使用偏移指针代替重复的字符串，域名偏移量固定12字节
            res[16 + self.name_length] = 0xc0
            res[17 + self.name_length] = 0x0c
            #自定义FLAGS、资源记录数量等
            res[18 + self.name_length] = 0x00
            res[19 + self.name_length] = 0x01
            res[20 + self.name_length] = 0x00
            res[21 + self.name_length] = 0x01
            res[22 + self.name_length] = 0x00
            res[23 + self.name_length] = 0x00
            res[24 + self.name_length] = 0x0d
            res[25 + self.name_length] = 0x34
            res[26 + self.name_length] = 0x00
            res[27 + self.name_length] = 0x04
            #利用.将ip分割为四个字段，转化为int型，存入res中，返回bytes型
            ip_ = ip.split('.')
            ip_1 = int(ip_[0])
            ip_2 = int(ip_[1])
            ip_3 = int(ip_[2])
            ip_4 = int(ip_[3])
            res[28 + self.name_length] = ip_1
            res[29 + self.name_length] = ip_2
            res[30 + self.name_length] = ip_3
            res[31 + self.name_length] = ip_4
            return bytes(res)
        else:
            #如果是无效的ip地址，则将前一种情况的flags更改为0x8583，指示名字错误，
            res = bytearray(32 + self.name_length)
            res[0] = self.ID >> 8
            res[1] = self.ID % 256
            res[2] = 0x85
            res[3] = 0x83
            res[4] = self.QDCOUNT >> 8
            res[5] = self.QDCOUNT % 256
            res[6] = 0x00
            res[7] = 0x01
```

```
            res[8] = self.AUTHOR >> 8
            res[9] = self.AUTHOR % 256
            res[10] = self.ADDI >> 8
            res[11] = self.ADDI % 256
            for i in range(12, 16 + self.name_length):
                res[i] = self.data[i]
            res[16 + self.name_length] = 0xc0
            res[17 + self.name_length] = 0x0c
            res[18 + self.name_length] = 0x00
            res[19 + self.name_length] = 0x01
            res[20 + self.name_length] = 0x00
            res[21 + self.name_length] = 0x01
            res[22 + self.name_length] = 0x00
            res[23 + self.name_length] = 0x00
            res[24 + self.name_length] = 0x0d
            res[25 + self.name_length] = 0x34
            res[26 + self.name_length] = 0x00
            res[27 + self.name_length] = 0x04
            ip_ = ip.split('.')
            ip_1 = int(ip_[0])
            ip_2 = int(ip_[1])
            ip_3 = int(ip_[2])
            ip_4 = int(ip_[3])
            res[28 + self.name_length] = ip_1
            res[29 + self.name_length] = ip_2
            res[30 + self.name_length] = ip_3
            res[31 + self.name_length] = ip_4
            return bytes(res)


if __name__ == '__main__':
    cache_file = 'example.txt'
    name_server=('223.5.5.5',53)
    relay_server = DNS_Relay_Server(cache_file,name_server)   #构造一个
DNS_Relay_Server实例
    relay_server.run() #运行
```

在程序中的处理

DNS query的处理：

```
#query内容解析
        """data -> name, querybytes, type, classify, len"""
        #初始化name为list，name length为0，name block为当前块name的长度
        self.name = []
        self.name_length = 0
        name_block = int(Msg_arr[12])
        #通过实验文档中所给的格式（长度+字符+长度+字符+……+\x0)解析name
        #当没有访问到\x0时，继续向后
```

```
        while(name_block != 0):
            #i为记录长度的counter
            i = 1;
            #如果当前不是第一part的name，先加一个"."，再加后面的part名字
            if( self.name_length != 0 ):
                self.name.append(".")
            #按照长度，访问该部分name
            while i <= name_block:
                self.name.append(chr(data[12 + self.name_length + i] ))
                i = i + 1
            #记录当前name长度
            self.name_length = self.name_length + name_block + 1
            #继续查看下一部分name的长度
            name_block = int(Msg_arr[12 + self.name_length])
        #将list类型转化为str类型，并解析剩下的信息
        self.name = ''.join(self.name)
        #记录当前name长度
        self.name_length = self.name_length + name_block + 1;
        #按照格式记录type和classify
        self.type = (Msg_arr[12 + self.name_length ] <<8) + Msg_arr[12 +
self.name_length + 1]
        self.classify = (Msg_arr[12 + self.name_length + 2] <<8) + Msg_arr[12 +
self.name_length + 3];
        #记录query部分总长度
        self.len = self.name_length + 4
        #记录总的数据
        self.data = data
```

DNS响应消息处理：

```
def generate_response(self,ip,Intercepted):
      #如果不是无效的ip，则按照格式生成响应报文
        if not Intercepted:
            #初始化res，并设定长度
            res = bytearray(32 + self.name_length)
            #ID
            res[0] = self.ID >> 8
            res[1] = self.ID % 256
            #FLAGS
            res[2] = 0x81
            res[3] = 0x80
            # 资源记录数量
            res[4] = self.QDCOUNT >> 8
            res[5] = self.QDCOUNT % 256
            # 回答记录数自定义为1
            res[6] = 0x00
            res[7] = 0x01
            res[8] = self.AUTHOR >> 8
            res[9] = self.AUTHOR % 256
            res[10] = self.ADDI >> 8
            res[11] = self.ADDI % 256
            #query内容解析
```

```
            for i in range(12, 16 + self.name_length):
                res[i] = self.data[i]
            #使用偏移指针代替重复的字符串，域名偏移量固定12字节
            res[16 + self.name_length] = 0xc0
            res[17 + self.name_length] = 0x0c
            #自定义FLAGS、资源记录数量等
            res[18 + self.name_length] = 0x00
            res[19 + self.name_length] = 0x01
            res[20 + self.name_length] = 0x00
            res[21 + self.name_length] = 0x01
            res[22 + self.name_length] = 0x00
            res[23 + self.name_length] = 0x00
            res[24 + self.name_length] = 0x0d
            res[25 + self.name_length] = 0x34
            res[26 + self.name_length] = 0x00
            res[27 + self.name_length] = 0x04
            #利用.将ip分割为四个字段，转化为int型，存入res中，返回bytes型
            ip_ = ip.split('.')
            ip_1 = int(ip_[0])
            ip_2 = int(ip_[1])
            ip_3 = int(ip_[2])
            ip_4 = int(ip_[3])
            res[28 + self.name_length] = ip_1
            res[29 + self.name_length] = ip_2
            res[30 + self.name_length] = ip_3
            res[31 + self.name_length] = ip_4
            return bytes(res)
        else:
            #如果是无效的ip地址，则将前一种情况的flags更改为0x8583，指示名字错误
            res = bytearray(32 + self.name_length)
            res[0] = self.ID >> 8
            res[1] = self.ID % 256
            res[2] = 0x85
            res[3] = 0x83
            res[4] = self.QDCOUNT >> 8
            res[5] = self.QDCOUNT % 256
            res[6] = 0x00
            res[7] = 0x01
            res[8] = self.AUTHOR >> 8
            res[9] = self.AUTHOR % 256
            res[10] = self.ADDI >> 8
            res[11] = self.ADDI % 256
            for i in range(12, 16 + self.name_length):
                res[i] = self.data[i]
            res[16 + self.name_length] = 0xc0
            res[17 + self.name_length] = 0x0c
            res[18 + self.name_length] = 0x00
            res[19 + self.name_length] = 0x01
            res[20 + self.name_length] = 0x00
            res[21 + self.name_length] = 0x01
            res[22 + self.name_length] = 0x00
            res[23 + self.name_length] = 0x00
```

```
            res[24 + self.name_length] = 0x0d
            res[25 + self.name_length] = 0x34
            res[26 + self.name_length] = 0x00
            res[27 + self.name_length] = 0x04
            ip_ = ip.split('.')
            ip_1 = int(ip_[0])
            ip_2 = int(ip_[1])
            ip_3 = int(ip_[2])
            ip_4 = int(ip_[3])
            res[28 + self.name_length] = ip_1
            res[29 + self.name_length] = ip_2
            res[30 + self.name_length] = ip_3
            res[31 + self.name_length] = ip_4
            return bytes(res)
```

程序的执行和输出

配置dns后的浏览器：无法显示知乎中的图片

程序运行

182.61.200.7 www.baidu.com
127.0.0.1 www.test1.com
可以直接通过程序创建响应报文返回

www.bilibili.com需要将报文转发出去后再转发给原地址

pic1.zhimg.com、pic2.zhimg.com为0.0.0.0，需要响应名字错误报文