# Project 3

# Student: Marcos Ondruska

# Student id: 268 5885

import sys

import random

from socket import socket, AF\_INET, SOCK\_DGRAM

def create\_query(id, domain\_name):

header = (id).to\_bytes(2, byteorder='big') # Transaction ID

header += (0x0100).to\_bytes(2, byteorder='big') # Flags (standard query)

header += (1).to\_bytes(2, byteorder='big') # Questions

header += (0).to\_bytes(2, byteorder='big') # Answer RRs

header += (0).to\_bytes(2, byteorder='big') # Authority RRs

header += (0).to\_bytes(2, byteorder='big') # Additional RRs

qname = b''

for part in domain\_name.split('.'):

qname += len(part).to\_bytes(1, byteorder='big')

qname += part.encode()

qname += (0).to\_bytes(1, byteorder='big') # End of QNAME

qtype = (1).to\_bytes(2, byteorder='big') # Type A

qclass = (1).to\_bytes(2, byteorder='big') # Class IN

return header + qname + qtype + qclass

def parse\_unsigned\_int(index, byte\_length, response):

num = int.from\_bytes(response[index: index + byte\_length], byteorder="big", signed=False)

return num, index + byte\_length

def parse\_name(index, response):

name = ''

initial\_index = index

jumped = False

while True:

length = response[index]

if length == 0:

index += 1

break

if (length & 0xC0) == 0xC0: # Pointer

if not jumped:

initial\_index = index + 2

jumped = True

pointer, \_ = parse\_unsigned\_int(index, 2, response)

index = pointer & 0x3FFF

else:

index += 1

name += response[index:index + length].decode('latin1') + '.'

index += length

if not jumped:

initial\_index = index

return name, initial\_index

def parse\_response(response):

index = 0

transaction\_id, index = parse\_unsigned\_int(index, 2, response)

flags, index = parse\_unsigned\_int(index, 2, response)

qdcount, index = parse\_unsigned\_int(index, 2, response)

ancount, index = parse\_unsigned\_int(index, 2, response)

nscount, index = parse\_unsigned\_int(index, 2, response)

arcount, index = parse\_unsigned\_int(index, 2, response)

# Skip the Question section

for \_ in range(qdcount):

name, index = parse\_name(index, response)

index += 4 # Skip QTYPE and QCLASS

answers = []

for \_ in range(ancount):

name, index = parse\_name(index, response)

rtype, index = parse\_unsigned\_int(index, 2, response)

rclass, index = parse\_unsigned\_int(index, 2, response)

ttl, index = parse\_unsigned\_int(index, 4, response)

rdlength, index = parse\_unsigned\_int(index, 2, response)

rdata = response[index:index + rdlength]

index += rdlength

if rtype == 1: # A record

ip = ".".join(map(str, rdata))

answers.append((name, ip))

authorities = []

for \_ in range(nscount):

name, index = parse\_name(index, response)

rtype, index = parse\_unsigned\_int(index, 2, response)

rclass, index = parse\_unsigned\_int(index, 2, response)

ttl, index = parse\_unsigned\_int(index, 4, response)

rdlength, index = parse\_unsigned\_int(index, 2, response)

rdata = response[index:index + rdlength]

index += rdlength

if rtype == 2: # NS record

ns, \_ = parse\_name(index - rdlength, response)

authorities.append((name, ns))

additionals = []

for \_ in range(arcount):

name, index = parse\_name(index, response)

rtype, index = parse\_unsigned\_int(index, 2, response)

rclass, index = parse\_unsigned\_int(index, 2, response)

ttl, index = parse\_unsigned\_int(index, 4, response)

rdlength, index = parse\_unsigned\_int(index, 2, response)

rdata = response[index:index + rdlength]

index += rdlength

if rtype == 1: # A record

ip = ".".join(map(str, rdata))

additionals.append((name, ip))

return answers, authorities, additionals

def resolve(domain\_name, root\_dns\_ip):

query\_id = random.randint(0, 65535)

query = create\_query(query\_id, domain\_name)

server\_ip = root\_dns\_ip

while True:

print(f"----------------------------------------------------------------")

print(f"DNS server to query: {server\_ip}")

sock = socket(AF\_INET, SOCK\_DGRAM)

sock.settimeout(5)

sock.sendto(query, (server\_ip, 53))

try:

response, \_ = sock.recvfrom(512)

except socket.timeout:

print(f"Timeout waiting for response from {server\_ip}")

return

sock.close()

answers, authorities, additionals = parse\_response(response)

print(f"Reply received. Content overview:")

print(f"{len(answers)} Answers.")

print(f"{len(authorities)} Intermediate Name Servers.")

print(f"{len(additionals)} Additional Information Records.")

print("Answers section:")

if answers:

for domain, ip in answers:

print(f"Name: {domain} IP: {ip}")

else:

print("No answers found.")

print("Authority Section:")

if authorities:

for domain, ns in authorities:

print(f"Name: {domain} Name Server: {ns}")

else:

print("No authorities found.")

print("Additional Information Section:")

if additionals:

for domain, ip in additionals:

print(f"Name: {domain} IP: {ip}")

else:

print("No additional information found.")

if answers:

print(f"----------------------------------------------------------------")

break

if not authorities:

print("No authorities found. Exiting.")

break

# Debugging: Print authorities and additionals

print("Authorities:")

for authority in authorities:

print(authority)

print("Additionals:")

for additional in additionals:

print(additional)

# Find the next server IP from the additional section or perform another query to resolve it

next\_server = None

for authority in authorities:

for additional in additionals:

if authority[1] == additional[0]:

next\_server = additional[1]

print(f"Matching NS record found: {authority[1]} -> {next\_server}")

break

if next\_server:

break

if not next\_server and authorities:

# If no IP in additional section, resolve the IP of the authoritative nameserver

next\_ns = authorities[0][1]

print(f"Resolving IP for the next nameserver: {next\_ns}")

temp\_query\_id = random.randint(0, 65535)

temp\_query = create\_query(temp\_query\_id, next\_ns)

sock = socket(AF\_INET, SOCK\_DGRAM)

sock.settimeout(5)

sock.sendto(temp\_query, (server\_ip, 53))

try:

temp\_response, \_ = sock.recvfrom(512)

temp\_answers, temp\_authorities, temp\_additionals = parse\_response(temp\_response)

if temp\_answers:

next\_server = temp\_answers[0][1]

print(f"IP resolved for {next\_ns}: {next\_server}")

except socket.timeout:

print(f"Timeout waiting for response from {server\_ip} while resolving {next\_ns}")

return

sock.close()

if next\_server:

print(f"Next DNS server to query: {next\_server}")

server\_ip = next\_server

else:

print("No additional info found. Exiting.")

break

if len(sys.argv) != 3:

print("Usage: python mydns.py domain-name root-dns-ip")

sys.exit(1)

domain\_name = sys.argv[1]

root\_dns\_ip = sys.argv[2]

resolve(domain\_name, root\_dns\_ip)