Queens College of CUNY
Department of Computer Science
Internet and Web Technologies (CSCI 355/655)
June 2024

Assignment #6:
"Network Addressing and Routing"
Due: JUne 18 , 2024

Overview:

In this assignment, we will use Python to perform various networking-related tasks, including ultimately obtaining the system's Routing Table and comparing it to a user-entered IPv4 address.

Submissions:

In the Google from, submit

- Assignment06.py (Python code)
- Assignment06.txt (console output).

Tasks:

Follow the guidelines and best practices from previous assignments with a Python programming component. Feel free to reuse code from previous assignments, in particular the one on socket programming.

[1] Write a function execute_command(cmd) to execute a windows shell command ("cmd").
See https://stackoverflow.com/questions/14894993/running-windows-shell-commands-with-python

```
# [1] Write a function execute_command(cmd) to execute a windows shell command ("cmd").
# See https://stackoverflow.com/questions/14894993/running-windows-shell-commands-with-python
def exec_cmd(cmd):
    return check_output(cmd, shell=True)
```

[2] Define a function get_routing_table by using the function from the previous step to run "route print" and then parsing the output into a table (2-D list).

```
# [2] Define a function get_routing_table by using the function from the previous step to run
"route print" and then parsing the output into a table (2-D list).
def get_routing_table(routing_data):
    s = routing_data.decode()
    s = s[s.find("Destination"): s.find("Internet6") - 1].strip()
    lines = s.split('\n')
    print (lines)
     data = [[get(x,0,17), get(x,18,37), get(x,38,44), get(x,45,62), get(x,63,70)] for x in
lines]
    for row in data:
        print (row)
    return data

def get(s,i,j):
    return s[i:j+1].strip()
```

[3] Prompt the user to enter an IPv4 address in dotted-decimal notation. If the IP address is 0 or empty, this is our cue to quit. Otherwise proceed to next step

[4] Define a function validate_address() to validate the address in two ways:

- Call your own function to check that the entered IP address is valid, that is, it consists of four decimal numbers, each between 0 and 255, and separated by dots (periods).
- Call code from socket package to validate it. See https://stackoverflow.com/questions/319279/how-to-validate-ip-address-in-python

[5] Define a function get_binary_address() to find the binary equivalent of an IP address in dotted decimal notation and use it on the inputted IP address.

[6] Define a function bitwise_and() to do the "bitwise-AND" of two bit-strings. You may either do it by iterating over the characters of the bit-strings, or use binary arithmetic. (See https://wiki.python.org/moin/BitwiseOperators)

[7] Define a function get_classful_address_type() to dmetermine its class A thru E. If it is class D or E, tell the user that D is for multicast or E is reserved, and return to step [2]. You can determine the class either directly from the dotted decimal notation or from the leading bits of the binary equivalent. (See https://en.wikipedia.org/wiki/Classful_network#Classful_addressing_definition)

[8] Define a function get_next_hop() to loop through the rows of the routing table from step [2] and determine the "Next Hop" for the user-inputted address. To determine which row is determinant, use the following algorithm:

- Do a bitwise-AND of the network mask with the destination IP address and see if you have a match with "Network Destination", the first column. Mathematically, this can be expressed at N = D & M where D is the Destination IP address, M is the (network) Mask, and N is the (destination) Network.
- Use the Metric column to decide between multiple matches (lowest value of Metric gets priority)
- If you have multiple matches and the Metric column is the same for all, then use the "Longest Prefix Match" to decide between ties, that is the one with more non-zero bits in the Mask.

[9] Compare the Next Hop to that provided by Python:

```
from pyroute2 import IPRoute
with IPRoute() as ipr:
     print(ipr.route('get', dst=ipAddress)
```

[10] Output the following information:

- The original inputted IP (destination) address in dotted-decimal notation
- The equivalent 32-bit address in binary notation
- The Class A, B, C, D, or E (the next two bullets apply only to classes A, B, C)
- The chosen Next Hop for that destination address
- How that address was chosen - match alone, metric, or longest-prefix match

[11] Repeat steps [3] thru [10] until the user enters nothing or just 0

Try your program on the following inputs. If you get tired of typing them in, you can instead read them in from an input file.

```
abc.def.ghi.jkl # not decimal
111-111-111-111 # not dotted
613.613.613.613 # format correct but numbers out of range
123.123.123 # valid numbers but only three octets
225.225.225.225 # valid address but class D
```

```
241.242.243.244 # valid address but class E
127.0.0.1 # valid - loopback address
52.3.73.91 # valid - an address for amazon
216.239.63.255 # valid - an address for google
```