IP Scanner 1.0 Documentation

A user manual to operating the IP Scanner software.

Table of Contents

Overview

The IP Scanner software was developed to speed up network scanning and to provide a quick overview of all connected ips on a network. The software utilizes the PING command and pings ips provided to it, getting a return value to determine if the ip is assigned to an actual device.

Software Requirements

The software was developed with a checklist of requirements that had to be filled. To release the software the following had to be included.

- Accept as input from a user, a network address in dotted decimal format to be scanned.

- Accept as input from a user, a start and end host number range to be scanned.

- Output must state which addresses are alive (have responded).

- Output must display to the user the format required for input before input becomes available.

- Carry out actions without errors.

- Include embedded comments to clarify the meaning of code for readability.

- Properly credited.

After these requirements were checked, the software was polished and released into version 1.0.

Algorithm

The created algorithm for the IP Scanner software was first designed as in code block 1, and then implemented as in code block 2.

Function is called, with target IP as the parameter.

A subprocess command is compiled, using the ping command with the parameter '-n 1' to send only a singular packet.

Once packet has been sent, await response, and if packet has been received, return to the call with True.

Code block 1.

```
Def ping(host):
    Command = ['ping', '-n', '1', host]
    Return subprocess.call(command) == 0
```

Code block 2.

Pseudo Code

Before the development of the software went ahead, the design team drafted a top-level view of the software. Code block 3 is the revised, reviewed and accepted pseudo code that best describes the software's code.

Program initialized.

Main function is called.

User prompted to enter in network IP.

User prompted to enter in start host.

User prompted to enter in end host.

Network IP is split into individual octets to have the last octet editable.

Timer is created to keep track of ping lifetime.

Loop through every number from start host to end host.

When a new number in the loop is hit, the last octet of the network IP is set to the new number.

The ping function then gets called with the network IP.

The ping function creates a command to ping the supplied network IP.

The ping function returns a thumbs up if the command successfully pings the network IP.

When all numbers from start host to end host have been hit, the function will finish, displaying the timer.

The program then asks the user if they want to quit the program or run another scan.

Code block 3.

Expected Running Results

The expected results when running the software is defined below in code blocks 4, 5 and 6.

cd [app.py source location]

py app.py

Code block 4 – How to launch the software.

Inputting a network address: xxx.xxx.xxx.0 (192.168.1.0)

Inputting a start host: x (40)

Inputting an end host: x (124)

Code block 5 – How to input variables and format.

>>> Active device found at: xxx.xxx.xxx.xxx [xx.xxxxxs] (192.168.1.14 [12.43145s])

Code block 6 – When an IP is found and is active.

Source Code

The following code block is the contents of the IP Scanner software. This is the raw source code and can be taken, placed in an empty python file and will run as expected.

```
#######################################################
#######################################################
######                                           ######
######                IP Scanner                 ######
######          Author: Daniel Americo           ######
######       Development Date: 28/09/2022        ######
######        Final Version Number: 1.0          ######
######                                           ######
######       This program is used to scan        ######
######         a network and get a list          ######
######             of all active IPs.            ######
######                                           ######
#######################################################
#######################################################

import subprocess, time, os

def cls(): # Helper function to clear the console
    os.system('cls' if os.name=='nt' else 'clear')

def logo():
    print(" _____ _____     _____                                ")
    print("|_   _|  __ \   / ____|                               ")
    print("  | | | |__) | | (___   ___ __ _ _ __  _ __   ___ _ __  ")
    print("  | | |  ___/   \___ \ / __/ _` | '_ \| '_ \ / _ \ '__| ")
    print(" _| |_| |       ____) | (_| (_| | | | | | | |  __/ |    ")
    print("|_____|_|      |_____/ \___\__,_|_| |_|_| |_|\___|_|    \n")

def ping(host):
    command = ['ping', '-n', '1', host] # Creates a string with the given
arguments to look like ("ping -n 1 host")
    return subprocess.call(command) == 0 # returns true if the ping is
successful

def main():
    cls()
    logo()
    print("Format IP: XXX.XXX.XXX.0")
```

```
    host = input("[*] Input local network IP: ")
    #host = "140.159.251.0" VU IP to test on, 192.168.56.0 does not work.
    try:
        startHost = int(input("[*] Input start host number: "))
    except ValueError: # Handles error if the user does not input a number
        main()
    try:
        endHost = int(input("[*] Input end host number: "))
    except ValueError: # Handles error if the user does not input a number
        main()
    print("\nScan started\n")

    octets = host.split('.') # Splits host IP into 4 octets to use later
    IP = octets[0] + "." + octets[1] + "." + octets[2] + "." # Add first 3
octets together to form almost complete IP

    timer = time.perf_counter() # Creates a timer

    for i in range(startHost, endHost + 1):
        res = ping(IP+str(i)) # Adds the last octet (i) to the IP, completing
it and then calling connect with the specified port (80)
        if res: # If we receive back True
            print(">>> Active device found at: " + IP+str(i) + f"
[{time.perf_counter() - timer:.5f}]") # Shows what address we were successful
in finding
    print(f"[{time.perf_counter() - timer:.5f}] Done Scanning") # Prints total
time, rounded to 5 decimal places (.5f)

    while True: # Allows the user to either do another scan or quit the
program
        checkControl = input("[S]can again | [Q]uit\n[*] Action: ")
        if checkControl == 's':
            main() # Calls the main function
        elif checkControl == 'q':
            quit() # Quits the program

if __name__ == "__main__":
    main()
```