گزارش پروژه شبکه های کامپیوتری

Ping

فاطمه آزادیان دلسم ۹۹۲۳۶۳۳

محمد مهدی دقیقی ۹۹۲۶۱۶۳

توضیح پیاده سازی

این پروژه از ۳ فایل اصلی **main** ، **ping** و **utils** تشکیل شده است . در بخش **ping** ، پیاده سازی ریز به ریز دستور **ping** انجام گرفته است . بخش **utils** به عنوان یک **validator** برای اطمینان از درستی ورودی برنامه عمل میکند و در نهایت بخش **main** که خط فرمان اصلی برنامه برای اجرا میباشد . در ادامه به تفصیل به جزئیات هر یک از فایل ها میپردازیم :

src/ping.py

اسکریپت **ping.py** یک جزء اصلی از برنامه میباشد که عملکرد ارسال درخواست‌های اکو **ICMP** (پینگ) به یک میزبان مشخص و دریافت پاسخ‌های اکو مربوطه را پیاده‌سازی می‌کند. این اسکریپت به طور نزدیک با لایه شبکه تعامل دارد، بسته‌های **ICMP** خام را تهیه، ارسال و پاسخ‌های ورودی را پردازش می‌کند.

```python
import os
import socket
import struct
import select
import time
import argparse
from utils import resolve_host, validate_ip, validate_hostname

# Constants
ICMP_ECHO_REQUEST = 8  # ICMP type code for echo request messages
DEFAULT_TIMEOUT = 1
DEFAULT_COUNT = 4
```

کتاب خانه ها و ماژول های لازم جهت استفاده در برنامه **import** شده است .

ثابت ها :

ICMP_ECHO_REQUEST: یک کد عددی که نوع "درخواست اکو **ICMP** "را نشان می‌دهد. برای نشان دادن اینکه بسته یک درخواست پینگ است استفاده می‌شود که در اینجا برای این منظور مقدار **8** را گرفته است .

DEFAULT_TIMEOUT: مقدار پیش فرض تایم اوت ست شده است (۱ ثانیه). این مقدار مدت زمانی میباشد که سرور صبر میکند تا پاسخ دریافت کند که اگر از این تایم عبور کند پیغام **"Timed out"** نشان داده میشود و به این معنا میباشد که پاسخی در زمان مشخص از **IP** یا هاست مربوطه دریافت نکرده است .

DEFAULT_COUNT: مقدار پیش فرض تعداد پینگ ها ست شده است که این مقدار در ورودی برنامه هم میتواند تغییذ کند .

```python
def checksum(source_string):
    sum = 0
    count_to = (len(source_string) // 2) * 2
    for count in range(0, count_to, 2):
        this_val = source_string[count + 1] * 256 + source_string[count]
        sum = sum + this_val
        sum = sum & 0xffffffff

    if count_to < len(source_string):
        sum = sum + source_string[-1]
        sum = sum & 0xffffffff

    sum = (sum >> 16) + (sum & 0xffff)
    sum = sum + (sum >> 16)
    answer = ~sum
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer
```

- **هدف** : محاسبه checksum داده‌های ورودی checksum .برای تضمین یکپارچگی داده‌ها در بسته ICMP استفاده می‌شود.
- **چگونگی کار** : داده‌های ورودی را در بخش‌های ۱۶ بیتی پردازش می‌کند، آن‌ها را جمع می‌زند و عملیات منطقی را انجام می‌دهد تا checksum را به دست آورد. محاسبه checksum بر اساس الگوریتم checksum استاندارد اینترنت انجام می‌شود.

```python
def receive_one_ping(sock, ID, timeout):
    time_remaining = timeout
    while True:
        start_time = time.time()
        readable = select.select( __rlist: [sock], __wlist: [], __xlist: [], time_remaining)
        time_spent = (time.time() - start_time)
        if readable[0] == []:  # Timeout
            return

        time_received = time.time()
        recv_packet, addr = sock.recvfrom(1024)

        icmp_header = recv_packet[20:28]
        type, code, checksum, packet_ID, sequence = struct.unpack(
            __format: "bbHHh", icmp_header
        )
        if packet_ID == ID:
            bytes_in_double = struct.calcsize("d")
            time_sent = struct.unpack( __format: "d", recv_packet[28:28 + bytes_in_double])[0]
            return time_received - time_sent

        time_remaining = time_remaining - time_spent
        if time_remaining <= 0:
            return
```

- **هدف** : دریافت پاسخ پینگ (بسته پاسخ اکو **ICMP**) از **socket**.
- **چگونگی کار** : منتظر بسته روی **socket** می‌ماند (از **select** برای انتظار تا **timeout** ثانیه استفاده می‌کند). وقتی یک بسته دریافت می‌شود، بررسی می‌کند که آیا بسته یک پاسخ اکو **ICMP** است و آیا با درخواست اکویی که ما فرستادیم مطابقت دارد (با تطابق **ID** و **checksum**). اگر پاسخ معتبر دریافت شود، زمان رفت و برگشت را با تفریق زمان ارسال درخواست (که در داده‌های بسته کدگذاری شده) و زمان فعلی محاسبه می‌کند.

```python
def send_one_ping(sock, dest_addr, ID):
    dest_addr = socket.gethostbyname(dest_addr)

    # Header is type (8), code (8), checksum (16), id (16), sequence (16)
    my_checksum = 0
    header = struct.pack( fmt: "bbHHh",  *v: ICMP_ECHO_REQUEST, 0, my_checksum, ID, 1)
    data = struct.pack( fmt: "d",  *v: time.time()) + bytes(48)

    # Calculate the checksum on the data and the dummy header.
    my_checksum = checksum(header + data)

    # Recreate the header with the correct checksum and pack the final packet
    header = struct.pack( fmt: "bbHHh",  *v: ICMP_ECHO_REQUEST, 0, socket.htons(my_checksum), ID, 1)
    packet = header + data
    sock.sendto(packet, (dest_addr, 1))
```

- **هدف** : ارسال یک بسته درخواست اکو **ICMP** به آدرس مقصد مشخص.
- **چگونگی کار** :
  - **ساخت هدر**: ایجاد هدر **ICMP** با نوع صحیح (۸ برای **ICMP_ECHO_REQUEST**)، کد (۰ برای درخواست اکو)، **checksum** (ابتدا ۰)، **ID** و شماره توالی.
  - **Payload/Data**: **Payload** را به بسته **ICMP** متصل می‌کند. **Payload** معمولا شامل زمان فعلی برای محاسبه زمان رفت و برگشت و برخی بایت‌ها برای پر کردن بسته به اندازه مورد نظر است.
  - **محاسبه Checksum**: محاسبه **checksum** هدر به علاوه داده و سپس دوباره وارد کردن این **checksum** به هدر.
  - **ارسال بسته**: بسته کامل (هدر + داده) به **dest_addr** با استفاده از یک **socket** خام فرستاده می‌شود.

```python
def do_one_ping(dest_addr, timeout=DEFAULT_TIMEOUT):
    icmp = socket.getprotobyname("icmp")
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
    except socket.error as e:
        if e.errno == 1:
            e.msg += " ICMP messages can only be sent from processes running as root."
            raise socket.error(e.msg)
    except Exception as e:
        print("Exception: {}".format(e))

    my_ID = os.getpid() & 0xFFFF

    resolved_addr = resolve_host(dest_addr)  # Resolve the hostname to an IP address
    if resolved_addr is None:
        return None

    send_one_ping(sock, resolved_addr, my_ID)
    delay = receive_one_ping(sock, my_ID, timeout)

    sock.close()
    return delay
```

هدف : اجرای یک فرآیند پینگ کامل: ارسال یک پینگ و انتظار برای پاسخ آن.

چگونگی کار : **send_one_ping** را برای ارسال یک درخواست اکو به **dest_addr** صدا می‌زند. **receive_one_ping** را برای گوش دادن به پاسخ اکوی مربوطه صدا می‌زند. زمان رفت و برگشت را اگر موفقیت‌آمیز باشد بر می‌گرداند یا اگر منقضی شود یا با خطا مواجه شود **None** بر می‌گرداند.

```python
def verbose_ping(dest_addr, count=DEFAULT_COUNT, timeout=DEFAULT_TIMEOUT):
    sent_pings = 0
    received_pings = 0
    total_time = 0.0

    for i in range(count):
        print(f"Pinging {dest_addr}...")
        sent_pings += 1
        try:
            delay = do_one_ping(dest_addr, timeout)
        except socket.gaierror as e:
            print(f"Failed. (socket error: '{e[1]}')")
            continue

        if delay is None:
            print(f"Ping Timed out. (timeout within {timeout}s)")
        else:
            delay_ms = delay * 1000
            total_time += delay_ms
            received_pings += 1
            print(f"Received ping in {delay_ms:.4f}ms")

    print(f"{sent_pings} packets transmitted, {received_pings} packets received.")

    if received_pings > 0:
        average_time = total_time / received_pings
        print(f"Average round-trip time: {average_time:.4f}ms")
        packet_loss = ((sent_pings - received_pings) / sent_pings) * 100
        print(f"Packet loss: {packet_loss:.2f}%")
    else:
        print("No response received.")
```

- **هدف** : ارسال چندین پینگ به یک میزبان و ارائه گزارش مفصل از نتایج.
- **چگونگی کار** : **count** یا همان تعداد پینگ‌ها را به **dest_addr** می‌فرستد، برای هر یک **do_one_ping** را صدا می‌زند. برای هر پینگ، چاپ می‌کند که آیا موفقیت‌آمیز بوده و زمان رفت و برگشت آن یا اگر منقضی شده است تایم اوت می دهد. پس از اتمام همه پینگ‌ها، اطلاعات آماری مانند تعداد بسته‌های فرستاده و دریافت شده، درصد از دست دادن بسته‌ها و میانگین زمان رفت و برگشت را محاسبه و چاپ می‌کند.

```python
def parse_arguments():
    parser = argparse.ArgumentParser(description='A simple Python ping command implementation.')
    parser.add_argument( *name_or_flags: 'destination', type=str, help='The IP address or domain to ping.')
    parser.add_argument( *name_or_flags: '-c', '--count', type=int, default=DEFAULT_COUNT, help='Number of pings to perform.')
    return parser.parse_args()
```

- **هدف** : تجزیه آرگومان‌های خط فرمان برای گرفتن ورودی‌های کاربر برای آدرس مقصد، تعداد پینگ‌ها و زمان انتظار.
- **چگونگی کار** : از کتابخانه **argparse** پایتون برای تعریف و تجزیه آرگومان‌های خط فرمان استفاده می‌کند. آرگومان‌های تجزیه شده را باز می‌گرداند که شامل آدرس مقصد (آدرس **IP** یا نام میزبان)، تعداد پینگ‌هایی که باید فرستاده شود (**count**) و زمان انتظار برای هر پینگ است.

در آخر هم تابع **main** وجود دارد که بتوان به صورت جدا به تست این فایل بدون نیاز به فایل **main** پرداخت. (بررسی اعتبار **IP** به کمک **utils.py**)

امتیازی : توانایی تنظیم صریح یک سرور **DNS** برای **resolve** دامنه توسط کتابخانه استاندارد **socket** در پایتون به طور مستقیم پشتیبانی نمی‌شود. تابع **socket.gethostbyname()** از تنظیمات **DNS** سیستم استفاده می‌کند ولی امکان مشخص کردن یک سرور **DNS** سفارشی درون کد را فراهم نمی‌آورد.

اگر بخواهیم توانایی تنظیم یک سرور **DNS** سفارشی را داشته باشیم، معمولاً باید از یک کتابخانه شخص ثالث که این قابلیت را فراهم می‌کند استفاده کنیم. یکی از کتابخانه‌های محبوب برای این منظور، **dnspython** است. با این حال، از آنجایی که نیازمندی‌های پروژه استفاده از کتابخانه‌های سطح بالاتر را محدود می‌کند و خواستار پایبندی به کتابخانه **socket** است، این قابلیت به طور مستقیم با محدودیت‌های داده شده قابل پیاده‌سازی نیست.

src/utils.py

اسکریپت **utils.py** یک ماژول کمکی در برنامه پینگ میباشد. این اسکریپت توابع کمکی را فراهم می‌کند که وظایف خاص و قابل استفاده مجدد مرتبط با اعتبارسنجی و حل داده‌های شبکه را انجام می‌دهند. این ماژول افزایش مدولاریتی و قابلیت استفاده مجدد کد در برنامه را فراهم میکند.

```python
def resolve_host(hostname):
    try:
        ip_address = socket.gethostbyname(hostname)
        return ip_address
    except socket.error as err:
        print(f"Cannot resolve hostname {hostname}: {err}")
        return None
```

- **هدف** : **resolve** یک نام دامنه (مانند **example.com**) به آدرس **IP** مربوطه.
- **چگونگی کار** : از تابع **socket.gethostbyname()** برای انجام **resolve DNS** استفاده می‌کند. این تابع نام دامنه قابل خواندن توسط کاربر را به یک آدرس **IP** عددی که تجهیزات شبکه می‌توانند درک کنند، ترجمه می‌کند.
- اگر **resolve** موفقیت‌آمیز باشد، آدرس **IP** را برمی‌گرداند.
- اگر **resolve** شکست بخورد (مثلاً اگر نام دامنه وجود نداشته باشد یا مشکل شبکه‌ای وجود داشته باشد)، **socket.error** را می‌گیرد، یک پیام خطا چاپ می‌کند و **None** را برمی‌گرداند.

```python
def validate_ip(ip):
    try:
        socket.inet_aton(ip)
        return True
    except socket.error:
        return False
```

- **هدف** : اعتبارسنجی اینکه آیا یک رشته یک آدرس **IPv4** معتبر است.
- **چگونگی کار** : از تابع **socket.inet_aton()** برای تلاش در تبدیل رشته به فرمت باینری فشرده ۳۲ بیتی (فرمت استاندارد آدرس**IPv4**) استفاده می‌کند.
- اگر تبدیل موفقیت‌آمیز باشد (به این معنی که **IP** معتبر است)، تابع **True**را برمی‌گرداند.
- اگر تبدیل شکست بخورد (که نشان‌دهنده یک **IP** نامعتبر است)، **socket.error** را می‌گیرد و **False** را برمی‌گرداند.

```python
def validate_hostname(hostname):
    if len(hostname) > 255 or len(hostname) == 0:
        return False
    allowed = re.compile( pattern: "(?!-)[A-Z\d-]{1,63}(?<!-)$", re.IGNORECASE)
    return all(allowed.match(x) for x in hostname.split("."))
```

- **هدف** : اعتبارسنجی اینکه آیا یک رشته یک نام میزبان به درستی تشکیل شده طبق قوانین معمول DNS است.
- **چگونگی کار** : بررسی می‌کند که آیا طول نام میزبان در محدوده معتبر است (0,255) اگر نه، **False** را برمی‌گرداند. از یک عبارت باقاعده برای اعتبارسنجی هر قسمت از نام میزبان استفاده می‌کند.

در نهایت برای تست این ماژول در تابع main تعدادی مقادیر تست نوشته شده است.

src/main.py

```python
def main():
    args = parse_arguments()

    if not validate_ip(args.destination) and not validate_hostname(args.destination):
        print("Invalid IP address or hostname.")
        return

    verbose_ping(args.destination, count=args.count)


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

در این تابع صرفا ورودی ها را در متغیر **args** قرار میدهد و پس از اعتبارسنجی آدرس ها به عملیات **ping** میپردازد.

نمونه ورودی / خروجی

```
PS D:\Mohammad\fake_ping\src> python main.py google.com -c 5
Pinging google.com...
Received ping in 20.7577ms
Pinging google.com...
Received ping in 31.2088ms
Pinging google.com...
Received ping in 46.9279ms
Pinging google.com...
Received ping in 31.2276ms
Pinging google.com...
Received ping in 34.9851ms
5 packets transmitted, 5 packets received.
Average round-trip time: 33.0214ms
Packet loss: 0.00%
```

```
PS D:\Mohammad\fake_ping\src> python main.py 192.168.1.150
Pinging 192.168.1.150...
Ping Timed out. (timeout within 1s)
Pinging 192.168.1.150...
Ping Timed out. (timeout within 1s)
Pinging 192.168.1.150...
Ping Timed out. (timeout within 1s)
Pinging 192.168.1.150...
Ping Timed out. (timeout within 1s)
4 packets transmitted, 0 packets received.
No response received.
```