

به نام خدا

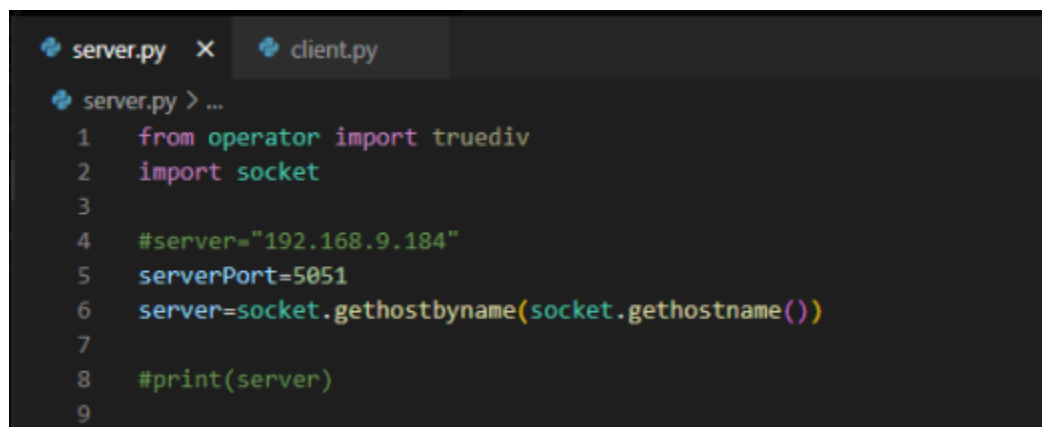
گزارش پروژه UDP Pinger

استاد: دکتر نادران طحان

اعضای تیم: زهرا معصومی 9873131

سارینا همت پور 9873136

در ابتدا فایل server.py را شرح می دهیم:



```
server.py X client.py
server.py > ...
1 from operator import truediv
2 import socket
3
4 #server="192.168.9.184"
5 serverPort=5051
6 server=socket.gethostbyname(socket.gethostname())
7
8 #print(server)
9
```

ابتدا کلاس های مورد نیاز برای برنامه را ایمپورت می کنیم.

سرور برای ارسال و دریافت پیام نیاز به شماره IP و شماره port دارد. Port# باید شماره ای رزرو نشده باشد که به این منظور مطابق تصویر در خط 5 شماره 5051 را به عنوان port# ست می کنیم.

شماره IP باید معادل با IP دستگاهی که سرور روی آن اجرا شده باشد. که در خط 4 برنامه این IP برای سرور set شده است.

اما می توان برای جلوگیری از hardcoding و امکان اجرای برنامه روی دیگر دستگاه ها بدون نیاز به ایجاد تغییر در کد، از دستور خط 6 استفاده کرد که در این برنامه با این دستور، IP کامپیوتری که سرور روی آن run می شود را به دست آورده ایم.

```

10 #socket
11 serverSocket=socket.socket(socket.AF_INET , socket.SOCK_DGRAM)
12
13 serverSocket.bind((server , serverPort))
14
15 print("The server is ready to recieve")
16

```

پس از آن برای انتقال اطلاعات، به یک socket از نوع UDP نیاز داریم.

برای ایجاد سوکت از دستور خط 11 استفاده می کنیم. در بخش اول ورودی socket.AF_INET نشان دهنده ورژن IPv4 و بخش دوم، تعیین کننده نوع سوکت است که برای ایجاد سوکت UDP، socket.SOCK_DGRAM را وارد کرده ایم.

حالا که سوکت ایجاد شده باید آن را با سرور bind کنیم که این کار سرور را برای on شدن و منتظر ماندن و شنیدن درخواست ها آماده می کند.

برای bind کردن نیز به آدرس سرور داریم که این آدرس شامل server IP و port# می شود. پس مطابق خط 13، با در نظر گرفتن ورودی های مورد نیاز، bind می کنیم.

پس از bind کردن، سرور برای شنیدن درخواست ها آماده است. پیامی برای اعلام این موضوع در خروجی برنامه چاپ می کنیم.

```

17 while True:
18     data, clientAddress=serverSocket.recvfrom(2048)
19     data=data.decode().upper()
20
21     #sendback
22     serverSocket.sendto(data.encode() , clientAddress)
23

```

برای این که سرور پس از دریافت یک پیغام و پاسخ به آن، همچنان قادر به شنیدن درخواست ها و ارسال پاسخ باشد، دستورها را در یک حلقه بی نهایت قرار می دهیم تا پس از انجام یک عملیات متوقف نشود و پس از بسته شدن هر کانکشن(توسط کلاینت) بتواند درخواست های جدید را قبول کند.

برای دریافت پیغام، سوکت باید پیغام(که محدودیت 2048bit برای آن در نظر گرفته شده) را از buffer بخواند(در خط 18)

این پیغام در متغیر data و آدرس کلاینت در متغیر clientAddress ذخیره می شود.(از آدرس ذخیره شده کلاینت برای ارسال پاسخ استفاده می شود).

سرور باید پاسخی به کلاینت ارسال کند. که در این مثال محتوای پاسخ ارسالی، uppercase شده ی پیام درخواست آن کلاینت است.

برای اعمال هرگونه تغییر در محتوای پیام(data) ابتدا نیاز به decode کردن آن پیام داریم. چرا که قبل از ارسال encode شده است.(خط 19)

حال برای ارسال پاسخ به کلاینت، از تابع sendto استفاده می کنیم(که برای این کار data را encode کرده و دیتای encode شده را به آدرسی از کلاینت که پس از دریافت درخواست ذخیره کرده بودیم، ارسال می کنیم). خط 22

توضیح فایل client.py

```
server.py client.py ×
client.py > ...
1  from encodings import utf_8
2  from http import server
3  import socket
4  import time
5
6
7  print("Starting ...")
8  #server="192.168.9.184"
9  server=socket.gethostbyname(socket.gethostname())
10 serverPort=5051
11 serverAddress=(server,serverPort)
12 seqNumber=0
13
```

آدرس سرور که شامل IP address و port# می شود باید به کلاینت اعلام شود. که برای این کار از یک tuple دوتایی که شامل این اطلاعات است استفاده کرده و آن را به نام serverAddress ذخیره می کنیم.

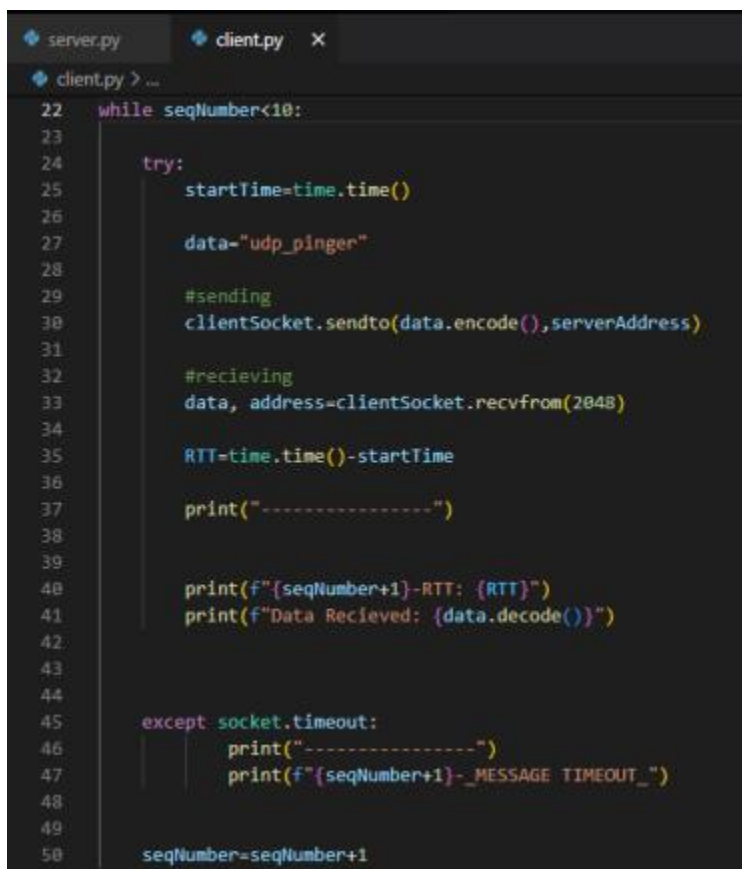
```

14 #socket
15 clientSocket=socket.socket(socket.AF_INET , socket.SOCK_DGRAM)
16 #clientSocket.bind((host , port))
17
18 clientSocket.settimeout(1)
19

```

کلاینت نیز مانند سرور برای ارسال و دریافت پیغام نیاز به سوکت دارد که در این بخش آن را ایجاد کرده ایم. مطابق توضیحات خواسته شده کلاینت تا 1 ثانیه پس از ارسال درخواست، در انتظار پاسخ سرور می ماند و اگر تا 1 ثانیه پاسخی دریافت نکرد، پیغام بعدی را ارسال می کند.

در خط 18 برای اعمال محدودیت سوکت، تایمری به مدت زمان 1 ثانیه set کرده ایم.



```

server.py  client.py  X
client.py > ...
22 while seqNumber<10:
23
24     try:
25         startTime=time.time()
26
27         data="udp_pinger"
28
29         #sending
30         clientSocket.sendto(data.encode(),serverAddress)
31
32         #recieving
33         data, address=clientSocket.recvfrom(2048)
34
35         RTT=time.time()-startTime
36
37         print("-----")
38
39         print(f"{seqNumber+1}-RTT: {RTT}")
40         print(f"Data Recieved: {data.decode()}")
41
42
43
44
45     except socket.timeout:
46         print("-----")
47         print(f"{seqNumber+1}-_MESSAGE TIMEOUT_")
48
49
50     seqNumber=seqNumber+1

```

کلاینت فقط اجازه ارسال 10 پیغام را دارد که برای این کار یک seq# تعریف کرده ایم که تعداد پیغام های ارسال شده از سمت کلاینت را می شمارد.

به علت محدودیت زمانی 1 ثانیه ای کلاینت برای دریافت پاسخ، دستورات مورد نیاز در بازه زمانی مشخص شده (1ثانیه) در بخش try و در صورت timeout شدن دستورات بخش except اجرا می شوند.

برای محاسبه RRT باید زمان قبل از ارسال (خط 25) و بعد از دریافت پیغام را ذخیره و RTT را به دست آوریم(خط 35).

برای ارسال پیغام دیتای ارسالی را encode کرده و با تابع sendto به آدرس سرور ذخیره شده می فرستیم و در انتظار دریافت پاسخ از سرور می شویم.

اگر پس از 1 ثانیه پاسخی دریافت نشد، پیغام MESSAGE TIMEOUT چاپ می کنیم(خط 47)

و اگر قبل از timeout شدن پاسخی از سمت سرور دریافت شد، اطلاعات خواسته شده شامل Seq# و RTT و دیتای دریافت شده را چاپ می کنیم.(خط 40-41)

این پروسه ارسال و دریافت پیغام تا 10 بار تکرار می شود و پس از آن کلاینت کانکشن را قطع می کند و اتصال به پایان می رسد:

```
51
52 print("Closing Socket")
53 clientSocket.close()
```

نمونه خروجی سمت کلاینت:

```
C:\Users\almas\Downloads\udp pinger project>python client.py
Starting ...
-----
1-RTT: 0.0
Data Recieved: UDP_PINGER
-----
2-RTT: 0.0
Data Recieved: UDP_PINGER
-----
3-RTT: 0.0
Data Recieved: UDP_PINGER
-----
4-RTT: 0.0
Data Recieved: UDP_PINGER
-----
5-RTT: 0.0009999275207519531
Data Recieved: UDP_PINGER
-----
6-RTT: 0.0
Data Recieved: UDP_PINGER
-----
7-RTT: 0.0
Data Recieved: UDP_PINGER
-----
8-RTT: 0.0
Data Recieved: UDP_PINGER
-----
9-RTT: 0.0009963512420654297
Data Recieved: UDP_PINGER
-----
10-RTT: 0.0
Data Recieved: UDP_PINGER
Closing Socket
C:\Users\almas\Downloads\udp pinger project>
```

نمونه خروجی اجرای برنامه:

خروجی سمت سرور:

```
PS C:\Users\almas\Downloads\udp pinger project>python server.py
The server is ready to recieve
█
```

خروجی سمت کلاینت:

```
C:\Users\almas\Downloads\udp pinger project>python client.py
Starting ...
-----
1-RTT: 0.0
Data Recieved: UDP_PINGER
-----
2-RTT: 0.0
Data Recieved: UDP_PINGER
-----
3-RTT: 0.0
Data Recieved: UDP_PINGER
-----
4-_MESSAGE TIMEOUT_
-----
5-RTT: 0.000997781753540039
Data Recieved: UDP_PINGER
-----
6-RTT: 0.0
Data Recieved: UDP_PINGER
-----
7-RTT: 0.0
Data Recieved: UDP_PINGER
-----
8-RTT: 0.0009999275207519531
Data Recieved: UDP_PINGER
-----
9-RTT: 0.0
Data Recieved: UDP_PINGER
-----
10-RTT: 0.0
Data Recieved: UDP_PINGER
Closing Socket
```