مستندات پروژه ی برنامه نویسی شبکه

محمدرضا برازش

91521059

بررسی کد سرور

ایجاد دو عدد لاک برای جلوگیری از ددلاک

```
L = threading.Lock()
L2 = threading.Lock()
CLIENTS = {}
HOST = ''
```

تعریف کردن سوکت از نوع tcp در پایتون

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST,8888))
s.listen(10)
COUNT = 0
chain = []
```

رشته ی زیر در موقع دریافت پیغام از هر کلاینت اجرا میشود و بر اساس پیغام گرفته شده دستوراتی را انجام میدهد

```
def clientThread(c,a):
    c.sendall(b"CONNECTED !")
    global COUNT
    while True :
        # try:
        d = c.recv(1024)
        print(d)
        request = d.decode(encoding="ASCII")
```

عملیات ثبت نام کردنیک کلاینت با دستور #REG

```
        if(request.startswith("REG#")):
            this = request[4:]
            CLIENTS[request[4:]] = [a[0],c]
            replay = b"REG#OK"
            c.sendall(replay)
```

عملیات خروج کلاینت از شبکه

```
        elif(request=="BYE"):
            replay = b"BYE#OK"
            c.sendall(replay)
            c.close()
```

دراین قسمت با گرفتن دستور peercheck سرور اقدام به چک کردن تمامی کلاینت هامیکند و از آنها میپرسد که مایل به دریافت فایل هستند یا خیر.

```
    elif(request.startswith("PEERCHECK#")):
        streamName = request[10:]
        permission = False
        # noTransmit = []
        # chain = []
        sender_name = None
        for name in CLIENTS.keys() :
            if CLIENTS[name][1] == c :
                permission = True
                sender_name = name
        if permission :
            chain.append(sender_name)
            for name in CLIENTS.keys():
                if CLIENTS[name][1] != c :
                    ask_ready = bytes("WANT?#" + streamName , encoding="ASCII")
                    CLIENTS[name][1].sendall(ask_ready)
        else:
            replay = b"STREAMREG#NOK#You are not registered !"
            c.sendall(replay)
```

در صورت پاسخ به سرور یک عدد به کانتر اضافه میشود . وقتی این کانتر به تعداد کلاینت ها منهای یک رسید یعنی همه ی کلاینت های گیرنده جواب خود را داده اند . پس جواب #peers#ready به کلاینت فرستنده فرستاده میشود.

```
    elif(request.startswith("PEER#YES")):
        L.acquire()
        for name in CLIENTS.keys():
            if CLIENTS[name][1] == c :
                chain.append(name)
                COUNT = COUNT + 1
                print(name,len(chain),COUNT )
                if COUNT == len(CLIENTS)-1:
                    replay = b"PEERS#READY#"
                    CLIENTS[chain[0]][1].sendall(replay)
        L.release()
    elif(request.startswith("PEER#NO")):
        L.acquire()
        for name in CLIENTS.keys():
            if CLIENTS[name][1] == c :
                COUNT = COUNT + 1
                print(name,len(chain),COUNT )
                if COUNT == len(CLIENTS)-1:
                    replay = b"PEERS#READY#"
                    CLIENTS[chain[0]][1].sendall(replay)
        L.release()
```

با دریافت دستور stream#start# با استفاده از زنجیره ی ساخته شده ؛ اقدام به ارسال آیپی و پورت کلاینت بعدی به هر کلاینت میکنیم . همچنین پورت فعلی که کلاینت باید بر روی آن سوکت بسازد تا کلاینت قبلی قابل برقراری ارتباط باشد نیز فرستاده میشود .

```
        elif(request.startswith("STREAM#START#")):
            L2.acquire()
            port = 31750
            for i in range(0,len(chain)-1):
                FROM = CLIENTS[chain[i]]
                TO = CLIENTS[chain[i+1]]
                if i == 0 :
                    msg =
bytes("START#"+str(port)+"#",encoding="ASCII")+bytes(CLIENTS[chain[1]][0],encoding="AS
CII")
                    FROM[1].sendall(msg)
                elif i == len(chain)-2 :
                    msg = bytes("MID#"+str(port-
1)+"#"+str(port)+"#",encoding="ASCII")+bytes(TO[0],encoding="ASCII")
                    FROM[1].sendall(msg)
                    port += 1
                    msg = bytes("FINISH#"+str(port-1),encoding="ASCII")
                    CLIENTS[chain[-1]][1].sendall(msg)

                else:

                    msg = bytes("MID#"+str(port-
1)+"#"+str(port)+"#",encoding="ASCII")+bytes(TO[0],encoding="ASCII")
                    FROM[1].sendall(msg)

                port += 1
            L2.release()
            print(chain,"THE CHAIN ...")

        else:
            break
        # except:
        #     print("err")
        #     break
```

حلقه ی اصلی سرور که با هر بار قبول کردن یک کلاینت یک ترد جدید میسازد تا به آن پاسخگو باشد .

```
while 1 :

    conn,adr = s.accept()

    threading._start_new_thread(clientThread,(conn,adr,))
    #print(CLIENTS)
```

بررسی کد کلاینت

مقدار دهی اولیه و ساخت سوکت tcp برای ارتباط با سرور

```
ERVERIP = ("127.0.0.1",8888)



s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(SERVERIP)
```

عملیات ثبت نام کلاینت بر روی سرور

```
connect_r = s.recv(1024).decode(encoding="ASCII")
print(connect_r)
name = str(input("ENTER ID?"))
SENDREQ = bytes("REG#"+name,encoding="ASCII")
s.sendall(SENDREQ)


register_r = s.recv(1024).decode(encoding="ASCII")
print(register_r)
```

مشخص نمودن کارکرد کلاینت . منتظر دریافت فایل یا قصد ارسال فایل ؟

```
inp = str(input("1-STREAM A FILE  2-WAIT FOR STREAM"))

if inp == "1" :
    SENDREQ = bytes("PEERCHECK# "+str(input("enter stream name?")),encoding="ASCII")
    s.sendall(SENDREQ)
    check_r = s.recv(1024).decode(encoding="ASCII")
    print(check_r)
    SENDREQ = bytes("STREAM#START#",encoding="ASCII")
    s.sendall(SENDREQ)
    ip_r = s.recv(1024).decode(encoding="ASCII")
    cd = ip_r.split("#")
    totalByte = 0
```

شروع عمل ارسال فایل در صورتی که کلاینت ارسال کننده فایل بود :

```
    if cd[0] == "START":
        f = open("simul.png",'rb')
        input("Press any key to start the stream ...")

        data = f.read(1024)

        ss = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        while (data):
            if( ss.sendto(data, (cd[2], int(cd[1]))) ):
                data = f.read(1024)
                totalByte += 1024
                print(totalByte)
        f.close()
        ss.close()
```

تشخیص رسیدن فایل در صورتیکه کلاینت گیرنده باشد .

```
if inp == "2" :
    stream_r = s.recv(1024).decode(encoding="ASCII")
    if input(stream_r[6:]+" is available. download ? Y/N") == "y":
        inp2 = "PEER#YES"
    else:
        inp2 = "PEER#NO"

    s.sendall(bytes(inp2,encoding="ASCII"))

    ip_r = s.recv(1024).decode(encoding="ASCII")
    cd = ip_r.split("#")
    print(ip_r)
```

شروع ساختن سوکت ها udp بر اساس اطلاعات و پورت های دریافت شده از سرور

```
    if cd[0] == "FINISH":
        fs = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        fs.bind(('', int(cd[1])))
        f = open("simul2.png",'wb')
        t = time.time()
        totalByte = 0
        while 1:
            totalByte += 1024
            fs.settimeout(5)
            data,addr = fs.recvfrom(1024)
            f.write(data)
            print(totalByte)
            print("AVG DOWNLOAD RATE = "+str(totalByte/t) , "TOTAL BYTES", totalByte)
        f.close()
        fs.close()

    elif cd[0] == "MID":

        ms = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        ms.bind(('', int(cd[1]) ))
        mt = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        print(ms)
        totalByte = 0
        while 1:
            d = ms.recvfrom(1024)

            totalByte += 1024
            print(totalByte)
            mt.sendto(d[0],( cd[3], int(cd[2])) )
```

پرسش ها

مشکلات موجود در طراحی پروتکل ارتباطی این پروژه را تحلیل کرده و راهکار خود را برای آن پیشنهاد دهید

در پروتکل شبکه هیچ سیاست خاصی برای قسمت ارتباط با peerها و اطلاع از وضعیت آنها برای آمادگی دریافت فایل در نظر گرفته نشده .

به همین دلیل با تغییرات در پروتکل پیشنهادی این امر اصلاح شد .

ابتدا کلاینتی که میخواهد فایل ارسال کند یک دستور #PEERCHECK به سرور میفرستد . سرور با گرفتن این دستور یک دستور [stream name]#?READY به تمامی کلاینت های گیرنده میفرستد و به آنها اطلاع میدهد که فایل مورد نظر قابل دریافت است . هر کلاینت میتواند مشخص کند که مایل به دریافت فایل هست یا خیر . این امر با دستور #PEER#NO و #PEER#YES مشخص میشود که کلاینت آنها را به سرور میفرستد . سرور با استفاده از کلاینت هایی که مایل به دریافت فایل بوده اند یک زنجیره تشکیل داده و به کلاینت فرستنده ی فایل هنگامی که تمامی کلاینت های ثبت شده در شبکه وضعیت خود را مشخص کردند اعلام میکند که میتواند فایل را ارسال کند . این عمل با دستور #PEER#READY انجام میشود .

سپس سرور آیپی و پورت های کلاینت ها را بر اساس رنجیره ی تشکیل شده به آنها میفرستد . پس ازینکار با دریافت پاسخ از کلاینت ها به کلاینت ارسال فایل دستور #Stream#READY را میفرستد و کار ارسال فایل شروع میشود .

در صورتی که چند کلاینت درخواست ارسال جریان دادهای را به سرور ارایه بدهند بهترین راهکار پیشنهادی شما چیست ؟

همان پروتوکل قبلی قابل استفاده است با این تفاوت که کافیست برای تعداد بیشتر از یک فرستنده اولا شرایط ریس و ددلاک را در نظر گرفته و از سمافور استفاده کنیم.

همچنین باید برای هر استریم جداگانه یه سری پورت udp جدا مشخص شود که کار ارسال همزمان چندین فایل در یک کلاینت به طور هم زمان قابل انجام باشد .

پیشنهاد شما برای تشکیل بهینه یک زنجیره ارتباطی در این پروژه به صورتی که زمان ارسال کامل جریان به حداقل برسد، چیست ؟

قبل از تشکیل زنجیره همه ی کلاینت ها از کلاینت های دیگر پینگ بگیرند . هر کلاینتی که با کلاینت دیگر پینگ کمتری داشت ارتباط بین آنها سریعتر خواهد بود پس این زنجیره را طوری طراحی میکنیم که کلاینت های متوالی در آن کمترین پینگ ممکن را داشته باشند . (الگوریتم جیکسترا یا هیل کلایمبینگ)

هنگام ارسال جریان دادهای احتمال قطع ارتباط کلاینتها وجود دارد .پیشنهادی برای از دست نرفتن کامل زنجیره ارتباطی به وجود آمده طرح کنید ؟

همانطور که به یاد دارید داده ها را برای ارسال در udp به چانک های کوچکتر تقسیم کردیم . میتوان پروتکل را طوری تغییر داد که در هر بار ارسال هر چانک جدید قبل از ارسال آن ابتدا کلاینت بعدی در زنجیره را پینگ کند و اگر آنلاین بود به کار خود ادامه دهد در غیر این صورت این موضوغ را به سرور گزارش نموده و از سرور آیپی و پورت کلاینت بعدی را بگیرد ..

با افزایش اندازه بستههای ارسالی در زنجیره ارتباطی، زمان دریافت بستهها را در آخرین کلاینت بررسی نمایید.

با افزایش اندازه ی بسته زمان تاخیر در رسیدن به اخرین کلاینت بیشتر میشود زیرا اگر اندازه ی چانک را خیلی بزرگ کنیم محدود تاخیر های دیگری مانند تاخیر ورودی/خروجی و تاخیرات شبکه میشویم .