برنامه‌نویسی سوکت

پرهام الوانی

پاییز ۱۳۹۸

فهرست مطالب

۱ مقدمه

در این آموزش قصد داریم یک سرور و کلاینت با استفاده از زبان پایتون پیاده‌سازی کنیم. در این برنامه کلاینت پس از متصل شدن به سرور، متنی را به سرور ارسال می‌کند. سرور پس از دریافت متن، آن را دوباره به کلاینت ارسال می‌کند. کلاینت با دریافت دوباره متن، متن دریافت شده را چاپ کرده و اتصال را قطع می‌کند. به چنین ابزارهایی Echo می‌گویند که در گذشته از آن‌ها برای صحت‌سنجی ارتباط استفاده می‌شد.

۲ سرور Echo

کد سرور به شرح زیر است، هر بخش کد به تفکیک در ادامه توضیح داده می‌شود.

```
import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 1373         # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

در ابتدا یک سوکت ایجاد می‌کنیم. کد زیر برای ایجاد سوکت استفاده می‌شود:

```
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

ورودی اول تابع socket نوع آدرس‌دهی و ورودی دوم نوع انتقال داده است. در این مثال با انتخاب AF_INET نوع آدرس‌دهی را از نوع IPv4 انتخاب می‌کنیم. در صورتی که قصد داریم آدرس را به صورت IP نوع چهارم یا hostname استفاده کنیم باید این نوع از آدرس‌دهی را به عنوان ورودی به تابع بدهیم. SOCK_STREAM برای ارتباط بر روی پروتکل TCP می‌باشد. در صورتی که قصد داشته باشیم پروتکل لایه انتقال را تغییر دهیم از این ورودی استفاده می‌کنیم.

از انواع آدرس‌دهی دیگر می‌توان به AF_PACKET، AF_INET6، AF_CAN، AF_BLUETOOTH و ...اشاره کرد.

پس از ایجاد سوکت آدرس مربوطه را به آن تخصیص می‌دهیم. در اینجا آدرس داده شده به سوکت آدرسی است که سوکت اطلاعات را از آن دریافت و از طریق آن ارسال می‌کند. آدرس شامل دوبخش host و port است. host یک رشته است که مشخص می‌کند این عمل گوش دادن روی کدام کارت شبکه صورت بگیرد و قسمت port مشخص کننده‌ی پورتی است که سرور روی آن گوش می‌دهد. مقدار host با توجه به نوع آدرس مشخص شده می‌تواند به صورت IP نوع چهارم، hostname یا رشته خالی باشد. در صورتی که رشته خالی یا *"0.0.0.0"* به عنوان host تنظیم شود، سرور از تمام کارت های شبکه خود برای ارتباط گیری استفاده می‌کند. در نهایت با مشخص شدن این اطلاعات می‌خواهیم که سوکت در حالت قبول تقاضا قرار بگیرد.

پارامتر ورودی تابع listen که backlog نام دارد تعداد ارتباط‌های پذیرفته نشده‌ای را مشخص می‌کند که سیستم عامل پیش از نپذیرفتن ارتباط‌های جدید خواهد پذیرفت. این تعداد به برنامه شما اجازه می‌دهد تا پذیرفتن درخواست‌ها را به دلایل مختلف به تعویق بیاندازد.

```
s.bind((HOST, PORT))
s.listen(1)
```

پس از مشخص شدن اطلاعات آدرس، سوکت باید منتظر برقرای ارتباط بماند. برنامه پس از برخورد با کد زیر در این همان جا متوقف می‌شود و منتظر دریافت درخواست ایجاد یک ارتباط توسط کلاینت می‌ماند. پس از دریافت درخواست برقراری ارتباط، سرور با فراخوانی کد زیر درخواست را تایید می‌کند. پس از تایید درخواست، آدرس درخواست دهنده و ارتباط ایجاد شده در دو متغیر addr و conn ذخیره می‌گردد.

```
conn, addr = s.accept()
```

حالا که ارتباط برقرار شده است، امکان انتقال اطلاعات فراهم می‌باشد. سرور هر بار به اندازه مشخصی از داده را می‌خواند. در این نمونه کد در هر بار **حداکثر** مقدار ۱۰۲۴ بایت داده خوانده می‌شود. در عمل شما زمانی که به طراحی پروتکل مشغول هستید باید روشی برای مشخص کردن هر درخواست داشته باشید، مثلا می‌توانید هر درخواست را با کاراکتر "*n*" خاتمه دهید در این صورت می‌بایست آرایه دریافتی را برای پیدا کردن این کاراکتر جستجو کنید، در صورتی که این کاراکتر در آرایه شما موجود بود باقی آرایه را برای مرحله بعدی ذخیره کنید یا ممکن است که این کاراکتر را پیدا نکنید که در این صورت می‌بایست تا خواندن دوباره صبر کنید. در نظر داشته باشید که کتابخانه‌های سطح بالاتری برای ساده کردن این عملیات وجود دارند.

```
with conn:
    print('Connected by', addr)
    while True:
        data = conn.recv(1024)
        if not data:
            break
```

در اینجا ما از پروتکل خاصی پیروی نمی‌کنیم بنابراین پس از خواندن داده آن را دوباره به کلاینت باز می‌گردانیم.

```
conn.sendall(data)
```

۳ کلاینت Echo به وسیله‌ی Telnet

پس از ایجاد سرور نیاز است که یک کلاینت نیز ایجاد کنیم تا بتوانیم با سرور ارتباط برقرار کنیم. یکی از راه‌های ساده تست سرور پیش از پیاده‌سازی کلاینت استفاده از دستور telnet در سیستم عامل لینوکس می‌باشد. (این دستور در سایر سیستم‌عامل‌ها نیز وجود دارد.) ساختار ساده شده این دستور به شکل زیر است:

```
telnet [ip] [port]
```

دستور telnet یک ارتباط TCP با آدرس IP و پورت داده شده برقرار می‌کند. شما می‌توانید هر آنچه که می‌خواهید در این ارتباط نوشته و با فشردن دکمه‌ی enter آن را به مقصدتان ارسال کنید. در ادامه اطلاعات دریافتی از سرور نیز برای شما به نمایش درخواهند آمد.
دقت داشته باشید که سرور شما تنها یکبار عملیات accept را انجام می‌دهد بنابراین بعد از بسته شدن ارتباط اولین کلاینت شما سرور شما به اتمام می‌رسد. برای جلوگیری از این مورد فراخوانی تابع accept عموما در یک حلقه بی‌نهایت قرار می‌گیرد.

```
import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 1373         # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    while True:
      conn, addr = s.accept()
      with conn:
          print('Connected by', addr)
          while True:
              data = conn.recv(1024)
              if not data:
                  break
              conn.sendall(data)
```

در این صورت سرور شما هرگز بسته نخواهد شد و شما می‌توانید به دفعات به آن متصل شوید.

۴ کلاینت Echo

```
import socket

HOST = '127.0.0.1'  # The server's hostname or IP address
PORT = 1373         # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall('      '.encode())
    data = s.recv(1024)
    message = data.decode()
    print(f'received {message!r}')
```

مانند سرور در این بخش نیز نیاز است ابتدا یک سوکت با ویژگی های یکسان با سوکت ساخته شده در سرور بسازیم:

```
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

پس از ایجاد سوکت، آدرس سرور را به سوکت داده تا درخواست ارتباط به سرور ارسال گردد. کد در این خط باقی می ماند تا زمانی که درخواست ارتباط توسط سرور قبول، رد و یا منقضی گردد. در صورت قبول نشدن درخواست برنامه خطا داده و متوقف می‌گردد. حتما در نظر داشته باشید که در هنگام استفاده از این سوکت در نرم افزارها، خطاهای بوجود آمده به علت عدم برقراری ارتباط به برنامه اصلی شما صدمه ای نزند. در صورت تایید ارتباط، برنامه ادامه می‌یابد.

```
s.connect((HOST, PORT))
```

برای رسیدگی به خطاها در نظر داشته باشید که می‌توان از ساختار Try/Except در پایتون استفاده کرد:

```
try:
    with open('file.log') as file:
        read_data = file.read()
except:
    print('Could not open file.log')
```

```
try:
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
```

کد زیر رشته Hello World را به سرور ارسال می‌کند. در نظر داشته باشید که استفاده از تابع encode به شما اجازه می‌دهد که رشته‌های UTF-8 را به رشته‌ای از بایت‌ها تبدیل کرده و ارسال کنید.

```
s.sendall('Hello, world'.encode())
```

پس از ارسال داده مانند قسمت سرور منتظر دریافت داده‌ها می‌مانیم. دقت داشته باشید که در عمل یکی از طرفیت می‌بایست ارتباط را خاتمه دهد. مثلا می‌توانیم در اینجا در سمت کلاینت پس از مطمئن شدم از دریافت صحیح رشته ارتباط را ببندیم یا مثلا کاربر با دستور Ctrl + C کلاینت و ارتباط را ببندد.

```
while True:
  data = conn.recv(1024)
  if not data:
      break
```

۵ ارتباط مبتنی بر UDP

زبان پایتون این امکان را می‌دهد که بتوانید با استفاده از پروتکل UDP به پراکنده کردن داده بپردازید. در مثال پیشرو یک کلاینت UDP برای ارسال پیام‌های همه‌پخشی ایجاد شده است. این کلاینت به آدرس 0.0.0.0 و پورت 44444 بسته شده و از آن برای دریافت بسته‌های UDP استفاده می‌کند. این کلاینت بسته‌های را به آدرس همه‌پخشی و پورت 37020 ارسال می‌کند. در نظر داشته باشید که در هردو این کلاینت‌ها ما تنظیمات مربوط به استفاده از وضعیت همه‌پخشی را انجام داده‌ایم.

```
import socket
import time

server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
server.settimeout(0.2)
server.bind(("", 44444))
message = b"your very important message"
while True:
    server.sendto(message, ('255.255.255.255', 37020))
    print("message sent!")
    time.sleep(1)
```

کلاینت دوم در ادامه به آدرس 0.0.0.0 و پورت 37020 بسته شده و داده‌ها را دریافت می‌کند. در اینجا هم مثل ارتباط TCP می‌بایست حداکثر داده‌ی دریافتی را مشخص کنیم ولی در نظر داشته باشید که اگر بسته‌ی دریافتی داده‌ی بیشتری از مقدار مشخص شده‌ی شما داشته باشد این داده‌ی اضافه **از بین خواهد رفت**.

```
import socket

client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP
client.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
client.bind(("", 37020))
while True:
    data, addr = client.recvfrom(1024)
    print("received message:",data,addr)
```

تفاوت اصلی میان ارتباط TCP و UDP در چگونگی ساخت سوکت و توابع ارسال و دریافت اطلاعات است. البته از شبکه به خاطر دارید که در ارتباط UDP یک اتصال شکل نمی‌گیرد و داده‌ها در قالب پیام به دست برنامه‌ی کاربردی می‌رسند. این پیام‌ها می‌توانند از ترتیب اولیه خود خارج شده باشند یا اینکه دچار خطا باشند.

نسخه‌ی اولیه این گزارش در بهار ۱۳۹۸ توسط سپهر صبور حاضر شده است. نسخه‌ی حاضر تنها شامل بهبودهایی اندک نسبت به نسخه‌ی اصلی می‌باشد. این سند برپایه بسته XƎPersian گونه 23.1 توسعه پیدا کرده است.