این مار خوش خط و خال

قسمت هفتم

برنامه‌نویسی به زبان پایتون

» احمد شریف پور

هرچند استفاده از سرویس‌های مبتنی بر شبکه‌های کامپیوتری، اینترنت، سیستم‌های اشتراک فایل و... پدیده‌هایی مدرن محسوب می‌شوند، اما ایده استفاده از منابع پردازشی و اطلاعاتی موجود در سایر کامپیوترهای یک شبکه، قدمتی به اندازه قدمت صنعت کامپیوتر دارد. نکته جالب توجه این‌که اصول انجام این کار نیز طی چندین دهه گذشته تغییر چندانی نکرده است. ایده برنامه‌های کلاینت/سرور یکی از قدیمی‌ترین روش‌های انجام این کار است که در این قسمت از مجموعه مقاله‌های آموزش پایتون به آن پرداخته‌ایم.

برای شروع کار باید بدانیم که اصولاً یک برنامه کلاینت/سرور چیست؟ برنامه‌های کلاینت/سرور در واقع دو برنامه مجزا هستند که به‌صورت همزمان اجرا شده و یکی از آن‌ها (سرور) اطلاعات یا منابعی را در اختیار برنامه دیگر (کلاینت) قرار می‌دهد. این دو برنامه می‌توانند در یک ماشین واحد اجرا شده یا در دو کامپیوتر مجزا در دو سوی متفاوت جهان به اجرا درآیند. به زبان ساده، هر زمانی که شما از یک برنامه (یا یک رابط وب) برای دسترسی به منابع یا داده‌های یک کامپیوتر یا یک برنامه دیگر استفاده‌می‌کنید، در حال کار با یک سیستم کلاینت/سرور هستید. نمونه ملموس چنین سیستمی زمانی است که از طریق یک کارت‌خوان خرید می‌کنید. در این هنگام دستگاه کارت‌خوان به عنوان یک کلاینت (که قصد انجام کاری را دارد) از یک سو و کامپیوترهای بانک (که نگه‌دارنده اطلاعات و انجام دهنده کار است) به عنوان سرور از سوی دیگر وارد عمل می‌شوند تا فرآیند خرید شما را تکمیل کنند. اگرچه فرآیندهایی نظیر این، بسیار پیچیده و حساس هستند و ما نخواهیم توانست چنین برنامه‌های پیچیده‌ای را ایجاد کنیم، اما می‌توانیم با مثال‌های ساده‌تر اصول کلی کار را دریابیم.

برای استفاده از یک سیستم کلاینت/سرور ابتدایی‌ترین کار، اتصال از ماشین کلاینت به ماشین سرور است. ما این کار را از طریق یک پایپ (pipe) یا یک سوکت (socket) انجام می‌دهیم. اگر در دوران کودکی با قوطی‌های خالی کنسرو، تلفن ساخته باشید، به سادگی می‌توانید اصول این کار را درک کنید. در این تلفن‌ها ارتباط بین دو قوطی خالی (سرور و کلاینت) از طریق یک رشته (اتصال یا connection) که به سوراخی در انتهای هر قوطی (سوکت) متصل است، برقرار می‌شود. ایده سوکت‌ها و اتصال بین کلاینت و سرور دقیقاً به همین شکل است. کلاینت اتصال (connection) مستقیمی به سرور دارد که این اتصال به سوکت معینی با شماره پورت مشخص در سرور متصل می‌شود.

برای ساختن برنامه کلاینت/سرور خودمان، ابتدا از سرور شروع می‌کنیم. در نخستین و ساده‌ترین تجربه ما، اتفاقاتی که در سرور رخ می‌دهد، در یک شبه کد ساده به صورت زیر خواهد بود:

- یک سوکت بساز
- نام ماشین میزبان سرور را بپرس
- یک پورت انتخاب کن
- سوکت را به کامپیوتر میزبان در پورت مشخص متصل کن
- منتظر یک اتصال باش
- اگر اتصال برقرار شد:
  - اتصال را قبول کن
  - دریافت اتصال را اعلام کن
  - اتصال را قطع کن

کد واقعی چنین سروری در فهرست ۱ آورده شده است.

```
1  #!/usr/bin/env python
2  #server1.py
3  import socket
4  soc=socket.socket()
5  hostname=socket.gethostname()
6  print “My hostname is “, hostname
7  port =21005
8  soc.bind((hostname,port))
9  soc.listen(5)
10 while True:
11     con,address = soc.accept()
12     print “I,m now connected to “ , address
13     con.send(“Hello and Goodbye”)
14     con.close()
```

فهرست۱ ساده‌ترین نمونه یک سرور

در خط ۳ ماجول سوکت را Import کرده‌ایم، سپس یک سوکت ساخته‌ایم. تابع ()socket که برای ساخت سوکت مورد استفاده قرار می‌گیرد، تعدادی آرگومان اختیاری دارد که کمی بعدتر به آن اشاره خواهیم کرد. پس از آن با کمک تابع ()gethostname نام ماشینی که این سرور را اجرا می‌کند، به دست آورده‌ایم. تابع ()bind که در خط ۸ از آن استفاده کرده‌ایم، آدرس و پورت موردنظر را به سوکت نسبت می‌دهد. در حالت پیش‌فرض (شبکه‌های مبتنی بر IPv4) آرگومان ورودی این تابع باید یک توپل دوتایی محتوای نام ماشین میزبان و شماره پورت (hostname,port) باشد. در انتها و در خط ۹ با تابع ()listen در انتظار برقراری یک اتصال نشسته‌ایم. سوکتی را که خود اقدام به برقراری ارتباط نمی‌کند و در انتظار اتصال سایر سوکت‌ها می‌ماند «سوکت گوش دهنده» یا Listener Socket می‌نامند. آرگومان تابع ()listen تعیین کننده حداکثر تعداد سوکت‌های در صف انتظار برای اتصال به سوکت «گوش دهنده» است و بیشترین مقدار ممکن برای آن به سیستم مورد استفاده بستگی دارد. پس از آن سرور در یک حلقه بی‌پایان، در صورت برقراری اتصال آن را می‌پذیرد (خط ۱۱)، دریافت اتصال را اعلام می‌کند (خط ۱۲)، پیغامی را به کلاینت ارسال کرده (خط ۱۳) و اتصال را قطع می‌کند (خط ۱۴). همان‌طور که مشاهده می‌کنید، خروجی تابع ()accept یک توپل دوتایی محتوای شیء سوکت متقاضی اتصال (con) و آدرس آن (address) است. ما برای ارسال اطلاعات و قطع ارتباط به این دو نیاز خواهیم داشت.

حال به یک کلاینت نیاز داریم تا پازل را تکمیل کند. در حالتی بسیار ساده این کلاینت همانند فهرست ۲ خواهد بود. این کد کاملاً شبیه کدهای سرور است با این تفاوت که، در این جا، به‌جای انتظار برای اتصال، سوکت ما اتصال را برقرار می‌کند و آنچه را که دریافت کرده است، چاپ کرده و اتصال را قطع می‌کند.

```
1  #!/usr/bin/env python
2  #clinet1.py
3
4  import socket
5  soc=socket.socket()
6  hostname=socket.gethostname()
7  port = 21005
8
9  soc.connect((hostname,port))
10 print soc.recv(1024)
11 soc.close
```

فهرست۲ ساده‌ترین نمونه یک کلاینت

برخلاف سوکت‌های گوش‌دهنده که باید توسط تابع ()bind به یک پورت یا آدرس بچسبند، سوکت‌هایی که قصد برقراری تماس را دارند، از تابع ()connect استفاده خواهند کرد. در خط ۹ مشاهده می‌کنید که تابع ()connect نیز همانند تابع ()bind از یک توپل محتوای نام میزبان و شماره پورت برای برقراری اتصال استفاده می‌کند. در خط ۱۰ نیز تابع ()recv نیز برای دریافت اطلاعات از اتصال برقرار شده مورد استفاده قرار می‌گیرد. در این تابع آرگومان داده شده اندازه بافر است که درواقع تعیین‌کننده حداکثر اندازه بلوک داده‌ای است که می‌تواند یک‌باره دریافت شود. این آرگومان برای هماهنگی بهتر با سخت‌افزارها و پروتکل‌های شبکه بهتر است توانی از ۲ مثلاً ۴۰۹۶ باشد.

توجه کنید که برای کار کردن این سیستم، باید هر دو برنامه به صورت همزمان به اجرا در بیایند. بنابراین، از دو پنجره مجزای کنسول یا دو نسخه مجزای محیط IDLE برای اجرای آن‌ها استفاده کنید. همچنین توجه داشته باشید که اگر در یک برنامه سوکت را نبندید، پورت به کار رفته

```
>>>
My hostname is  x2
I'm now connected to  ('192.168.1.2', 1439)
```

شکل۱ خروجی سرور شماره ۱

```
#!/usr/bin/env python
# server2.py
from socket import *
import sys
import os
BUFSIZ = 4096
HOST = ""
PORT = 29876
ADDR = (HOST,PORT)
class ServCmd:
    def __init__(self):
        self.serv = socket(AF_INET,SOCK_STREAM)
        self.serv.bind((ADDR))
        self.cli = None
        self.listeningloop  = 0
        self.processingloop = 0
        self.run()
    def run(self):
        self.listeningloop = 1
        while self.listeningloop:
            self.listen()
            self.processingloop = 1
            while self.processingloop:
                self.procCmd()
            self.cli.close()
        self.serv.close()
    def listen(self):
        self.serv.listen(5)
        print "Listening for Client"
        cli,addr = self.serv.accept()
        self.cli = cli
        print "Connected to ", addr
    def procCmd(self):
         cmd = self.cli.recv(BUFSIZ)
         if not cmd:
            return
         print "Received command: ", cmd
         self.servCmd(cmd)
         if self.processingloop:
            proc = os.popen(cmd)
            outp = proc.read()
            if outp:
                self.cli.send(outp)
            else   :
                self.cli.send("OK")
    def servCmd(self,cmd):
        cmd = cmd.strip()
        if cmd == "GOODBYE":
            self.processingloop = 0
if __name__ == "__main__":
    serv = ServCmd()
```

فهرست ۳ نسخه پیشرفته‌تر یک سرور

```
ahmad@Enilli:/media/Project/Shabakeh/121/Python-Part7$ python ./client2.py
total 113
drwxrwxrwx 1 root root  4096 2011-04-08 10:34 .
drwxrwxrwx 1 root root     0 2011-04-07 21:48 ..
-rwxrwxrwx 1 root root   184 2011-04-08 00:42 client1.py
-rwxrwxrwx 1 root root   724 2011-04-08 10:40 client2.py
-rwxrwxrwx 1 root root  2144 2011-04-08 02:07 Pic1.png
-rwxrwxrwx 1 root root 14333 2011-04-08 03:42 Pic2.png
-rwxrwxrwx 1 root root 59392 2011-04-08 11:33 Python7.doc
-rwxrwxrwx 1 root root 18236 2011-04-07 22:48 Python7.odt
-rwxrwxrwx 1 root root   343 2011-04-08 00:42 server1.py
-rwxrwxrwx 1 root root  1399 2011-04-08 02:34 server2.py

ahmad@Enilli:/media/Project/Shabakeh/121/Python-Part7$
```

شکل ۳ خروجی نسخه دوم برنامه‌های کلاینت و سرور

در آن حتی پس از بستن IDLE یا کنسول اجرا کننده، در اجراهای بعدی باز و قابل استفاده نخواهد بود. در چنین وضعیتی برای اجراهای بعدی، باید از اعداد پورت جدید، اما یکسان در هر دو سمت سرور و کلاینت استفاده کنید.

خروجی این برنامه‌ها کاملاً قابل پیش‌بینی است. در سمت سرور خروجی چیزی همانند شکل ۱ و در سمت کلاینت فقط عبارت Hello and Goodbye خواهد بود (نام میزبان یاشماره IP در سیستم شما به احتمال زیاد متفاوت خواهد بود).

همان‌طور که مشاهده کردید، پیاده‌سازی چنین نمونه محدودی بسیار ساده است. در مرحله بعدی قصد داریم سروری بسازیم که قادر به انجام کار مفیدی به جز سلام و خداحافظی باشد. کد این سرور را در فهرست ۳ مشاهده می‌کنید.

در خطوط ۶ الی ۹ و پس از import کردن ماجول‌های مورد نظر (در قسمت‌های بعدی درباره ماجول‌های os و sys بیشتر صحبت خواهیم کرد)، تعدادی متغیر را برای استفاده‌های بعدی مقداردهی کرده‌ایم. متغیر BUFSIZ اندازه بافر مورد استفاده را برای دریافت اطلاعات ورودی از سمت کلاینت تعیین می‌کند. همچنین ما پورتی را که روی آن به انتظار اتصال خواهیم بود، در خط ۸ تعریف کرده‌ایم و پس از آن توپلی را برای نگه‌داری نام کامپیوتر میزبان و شماره پورت به‌وجود آورده‌ایم.

سپس در خط ۱۰ کلاسی با نام ServCmd تعریف کرده‌ایم و در تابع ()__init__ سوکت را ساخته و آن را به نام و آدرس فعلی متصل می‌کنیم. در خط ۱۲ مشاهده می‌کنید که این بار برای ساخت سوکت دو آرگومان به آن ارسال کرده‌ایم.

همان‌گونه که پیش‌تر نیز اشاره کردیم، تابع سازنده سوکت چهار آرگومان عددی اختیاری دارد که نخستین آن‌ها تعیین کننده نوع آدرس‌دهی سوکت است. زیرا نحوه آدرس‌دهی در شبکه‌های یونیکس، IPv4 و IPv6 با یکدیگر متفاوت است. حالت پیش‌فرض این آرگومان مقدار عددی ۲ یا شبکه‌های مبتنی بر IPv4 است. آرگومان دوم تعیین کننده نوع خود سوکت است و در واقع بیان کننده فرمت انتقال اطلاعات روی اتصال ایجاد شده توسط این سوکت است. حالت پیش‌فرض آن عدد ۱ یا انتقال اطلاعات از طریق جریان (Stream) است. آرگومان‌های سوم و چهارم فراتر از مجال این مقاله هستند. در مثال این قسمت به جای آرگومان‌های عددی برای خواناتر بودن کد، از ثابت‌هایی استفاده شده که توسط خود ماجول Socket در دسترس قرار می‌گیرند. این ثابت‌های عددی، یعنی AF_INET و SOCK_STREAM توسط ماجول socket تعریف و مقداردهی می‌شوند. به عبارت دیگر، دستور خط ۱۲ در واقع معادل self.serv=socket(2,1) خواهد بود.

```python
#!/usr/bin/env python
# client2.py

from socket import *
from time import time
from time import sleep
import sys
BUFSIZE=4096

class CmdLine:
    def __init__(self,host):
        self.HOST=host
        self.PORT=29876
        self.ADDR=(self.HOST,self.PORT)
        self.sock=None
    def makeConnection(self):
        self.sock=socket(AF_INET,SOCK_STREAM)
        self.sock.connect(self.ADDR)
    def sendCmd(self,cmd):
        self.sock.send(cmd)
    def getResult(self):
        data=self.sock.recv(BUFSIZE)
        print data

if __name__=="__main__":
    conn=CmdLine("localhost")
    conn.makeConnection()
    conn.sendCmd("ls -al")
    conn.getResult()
    conn.sendCmd("GOODBYE")
```

فهرست۴ نسخه پیشرفته‌تر یک کلاینت

سپس با تعریف تابع Run (در خط ۱۸) روی ورودی‌های سوکت موردنظر به انتظار اتصال و دریافت دستور خواهیم نشست. در خط ۲۷ تابعی برای شنود پورت و آدرس مورد نظر تعریف کرده‌ایم.

آدرس و شی کلاینت مربوط به سوکت متصل‌شونده را در خط ۳۰ و به کمک تابع ()accept دریافت کرده‌ایم. توجه کنید که برای در دسترس بودن شی سوکت متصل شونده در تمام قسمت‌های کد، در خط ۳۱ آن را به یکی از خاصیت‌های شی ServCmd تبدیل کرده‌ایم. در خط ۳۳ تابعی برای پردازش دستورات دریافتی ایجاد شده است.

در بخش اول این تابع وجود یا عدم وجود ورودی را چک کرده‌ایم. اگر دستوری دریافت شده باشد آن را به تابع ()servCmd ارسال می‌کنیم. به تفاوت میان ServCmd که یک کلاس است با تابع ()servCmd که در همان کلاس تعریف شده است دقت کنید.

این تابع به کمک متد ()strip در خط ۴۷ که مختص نوع داده رشته است، فضاهای خالی احتمالی در ابتدا و انتهای رشته را حذف کرده و دستور را بصورت خالص بازمی‌گرداند. همچنین در صورت دریافت

```python
if __name__ == "__main__":
        conn=CmdLine('localhost')
        conn.makeConnection()
        loop=1
        while loop:
                cmd=raw_input("Enter a command:")
                if  cmd=="finish:
                        loop=0
                else:
                        conn.sendCmd(cmd)
                        con.getResult()
```

فهرست۵ کد جایگزین برای حفظ ارتباط کلاینت و سرور

دستور GOODBYE چرخه اجرای برنامه را متوقف می‌کند. هنگامی که دستوری را از جانب کلاینت دریافت کردیم، از تابع ()popen متعلق به ماجول os برای پردازش آن استفاده می‌کنیم (خط ۴۰). این تابع در واقع یک پوسته خط فرمان ایجاد کرده و دستور دریافتی را در آن اجرا می‌کند.

پس از اجرای دستور خروجی حاصل از آن را در خط ۴۱ دریافت کرده و در خط ۴۲ کنترل می‌کنیم. درصورتیکه دستور دارای خروجی باشد، آن را به سوکت برقرار کننده اتصال بازمی‌گردانیم و در غیر این صورت پیغام OK بازگردانده می‌شود (خطوط ۴۲ تا ۴۵).

حال نوبت کدهای سمت کلاینت است که آن‌ها را در فهرست ۴ مشاهده می‌کنید. این کدها بسیار ساده‌تر از کدهای سمت سرور هستند. در اینجا از همه قسمت‌ها به جز بخش ارسال دستور صرف‌نظر می‌کنیم، زیرا خود شما دیگر می‌توانید آن‌ها را تجزیه و تحلیل کنید. در قسمت ارسال دستور و در خط ۲۳۴ دستور ساده ls را برای دریافت فهرست فایل‌های موجود در سمت سرور اجرا کرده‌ایم (در صورت استفاده از ماشین‌های ویندوزی باید آن را به dir تغییر دهید).

نتیجه اجرای این دو برنامه در ماشین من همانند شکل ۳ است. برای برقراری ارتباط بین ماشین‌های متفاوت در یک شبکه کافی است در فایل مربوط به کلاینت، در خط ۵۳۲۴۵ عبارت localhost را به نام یا آدرس IP ماشینی که برنامه سرور را اجرا می‌کند، تغییر دهید. به این ترتیب، امکان انتقال اطلاعات بین ماشین‌های مختلف را در اختیار خواهیم داشت.

اگر می‌خواهید برنامه کلاینت شما ارتباط با سرور را قطع نکند و بتوانید دستورات دیگری را نیز با این سیستم امتحان کنید، کافی است خطوط ۲۵ تا ۳۰ فایل کلاینت را با قطعه کدهای فهرست ۵ تعویض کنید. در این صورت در یک حلقه دائمی تا زمانی که شما دستور finish را صادر نکرده باشید، دستورات جدید از شما پرسیده شده و روی سرور اجرا خواهند شد.

با این مثال‌های ساده شما به‌صورت عملی با مفهوم برنامه‌های کلاینت/سرور آشنا شده‌اید و می‌توانید برنامه‌هایی را به‌صورت توزیع شده روی کامپیوترهای شبکه خود به اجرا درآورید. در بخش‌های بعدی مجموعه و با تکمیل مباحث مربوط به تعامل پایتون و سیستم‌عامل بهتر خواهید توانست قابلیت‌های سیستم‌های کلاینت/سرور را مورد استفاده قرار دهید.