

Primeri iz apstrakcije klijenti-slušaoči u paketu “multiprocessing”

Apstrakcija klijent-slušaoči odgovara klasičnoj TCP/IP klijent-server arhitekturi (npr. Web klijent-server, itd.), s tim što dodatno obezbeđuje zaštićenu vezu klijent-slušaoč na bazi autentifikacije.

Osnovne klase vezane za apstrakciju klijenti-slušaoči su: Client i Listener.

Glavna referenca je Python online dokumentacija:

<https://docs.python.org/3/library/multiprocessing.html>

Klijenti-slušaoči

Primer 1: Kod za prvi primer je u modulima ep1_listener.py i ep1_client.py. Ova dva modula se pokreću u dva odvojena terminala.

Modul ep1_listener.py: Proces main prvo napravi objekat listener (pomoću konstruktora Listener), i prihvati vezu sa klijentom (pomoću metode accept). Nakon toga, ispiše adresu klijenta, i pošalje mu tri poruke pomoću dve varijante metode send: (1) send šalje listu, i (2) send_bytes šalje string i niz.

Tabela 1.a: Programski kod modula ep1_listener.py.

```
from multiprocessing.connection import Listener
from array import array

address = ('localhost', 6000) # family is deduced to be 'AF_INET'

def main():
    with Listener(address, authkey=b'secret password') as listener:
        with listener.accept() as conn:
            print('connection accepted from', listener.last_accepted)

            conn.send([2.25, None, 'junk', float])

            conn.send_bytes(b'hello')

            conn.send_bytes(array('i', [42, 1729]))

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

Tabela 1.b: Rezultat izvršenja modula ep1_listener.py.

```
D:\Z>python ep1_listener.py
connection accepted from ('127.0.0.1', 59675)
```

Modul ep1_client: Proces main prvo napravi objekt veze conn (pomoću konstruktora Client), a zatim primi tri poruke od servera, pomoću tri varijante metode recv: (1) recv prima listu, (2) recv_bytes prima string, i (3) recv_bytes_into prima niz i upisuje ga u zadati niz.

Tabela 1.c: Programski kod modula ep1_client.py.

```
from multiprocessing.connection import Client
from array import array

address = ('localhost', 6000)

def main():
    with Client(address, authkey=b'secret password') as conn:
        print(conn.recv())          # => [2.25, None, 'junk', float]

        print(conn.recv_bytes())   # => 'hello'

        arr = array('i', [0, 0, 0, 0, 0])
        print(conn.recv_bytes_into(arr)) # => 8
        print(arr)                  # => array('i', [42, 1729, 0, 0, 0])

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

Tabela 1.d: Rezultat izvršenja modula ep1_client.py.

```
D:\Z >python ep1_client.py
[2.25, None, 'junk', <class 'float'>]
b'hello'
8
array('i', [42, 1729, 0, 0, 0])
```

Primer 2: Kod za drugi primer je u datoteci ep2_node_to_node.py. Ovaj program je potrebno pokrenuti kao dve različite instance, tj. u dva odvojena terminala, ali sa različitim vrednostima komandnih argumenata.

Ove dve instance programa glume dva različita čvora u mreži. Obe instance programa imaju proces klijenta koji izvršava funkciju main() i proces servera koji izvršava funkciju server_fun(). U jednom programu, klijent i server su povezani sa redom queue, u koji server upisuje upisuje poruke od udaljenog klijenta.

API za lokalnog klijenta čine funkcije `sendMsg` i `rcvMsg`. Funkcija `sendMsg` šalje poruku udaljenom serveru (njeni parametri su port udaljenog servera i poruka). Funkcija `rcvMsg` prima poruku od lokalnog servera (njen parametar je lokalni red queue).

Tabela 2.a: API između lokalnog klijenta i lokalnog servera.

```
def sendMsg(remote_server_address, msg):
    with Client(remote_server_address, authkey=b'Lets work together') as conn:
        conn.send(msg)

def rcvMsg(queue):
    return queue.get()
```

Lokalni server prvo postavi svoju adresu koju čini par (IP adresa, port), i napravi objekt listener pomoću konstruktor `Listener`. Zatim, u beskonačnoj petlji: (1) prihvata vezu sa udaljenim klijentom (metoda `accept`), (2) prima od njega poruku (metoda `recv`), (3) upisuje tu poruku u red (metoda `put`), i (4) prekida petlju u slučaju prijema specijalne poruke 'exit'.

Tabela 2.b: Lokalni server.

```
def server_fun(local_port, queue):
    # Set the address of the local node's server
    local_server_address = ('localhost', local_port)

    # Send fixed message
    with Listener(local_server_address, authkey=b'Lets work together') as listener:

        while True:
            with listener.accept() as conn:
                msg = conn.recv()

                # Forward msg to local node's process
                queue.put(msg)

                # Exit if msg is 'exit'
                if msg == 'exit':
                    break
```

Lokalni klijent – obrada komandne linije i inicijalizacija. Argumenti komandne linije su portovi lokalnog i udaljenog servera. Koraci inicijalizacije su: (1) napravi red queue, (2) napravi i pokreni proces lokalnog servera, (3) postavi adresu udaljenog servera (IP adresa, port), i (4) ispiši korisničko uputstvo za slanje poruka drugom čvoru mreže.

Tabela 2.c: Lokalni klijent – obrada komandne linije i inicijalizacija.

```
def main():
    # Parse command line arguments
    if len(sys.argv) != 3:
```

```

print('Program usage: ep2_node_to_node local_port remote_port, ...')
print('Note: This program expects that its peer instance swaps ports, ...')
exit()

# Set ports
local_port = int( sys.argv[1] )
remote_port = int( sys.argv[2] )

# Create queue for messages from the local server
queue = Queue()

# Create and start server process
server = Process(target=server_fun, args=(local_port,queue))
server.start()

# Set the address of the peer node's server
remote_server_address = ('localhost', remote_port)

# Send a message to the peer node and receive message from the peer node.
# To exit send message: exit.
print('Send a message to the peer node and receive message from the peer node.')
print('To exit send message: exit.')

```

Lokalni klijent – beskonačna petlja i završetak. Klijent u petlji: (1) učitava poruku sa tastature, (2) pošalje tu poruku udaljenom server (pomoću API funkcije `sendMsg`), (3) primi poruku od svog lokalnog server (pomoću funkcije `rcvMsg`), (4) istampa poruku, i (5) prekine petlju ako je primio poruku 'exit'.

Tabela 2.d: Lokalni klijent – beskonačna petlja i završetak.

```

while True:
    # Input message
    msg = input('Enter message: ')

    # Send message to peer node's server
    sendMsg(remote_server_address, msg)

    # Get message from local node's server
    msg2 = rcvMsg(queue)
    print('Message received: %s \n' % (msg2))

    if msg == 'exit':
        break

# Join with server process
server.join()

# Delete queue and server

```

<pre>del queue del server</pre>

Primer pokretanja dve instance programa ep2_node_to_node.py u dva terminala.

U sledeće dve tabele je dat rezultat izvršenja dve instance programa u dva različita terminala.

Tabela 2.e: Rezultat izvršenja modula ep2_node_to_node.py u prvom terminalu.
--

<pre>C:\Z\>python ep2_node_to_node.py 6000 6001 Send a message to the peer node and receive message from the peer node. To exit send message: exit. Enter message: hello Message received: hi Enter message: exit Message received: exit</pre>
--

Tabela 2.f: Rezultat izvršenja modula ep2_node_to_node.py u drugom terminalu.

<pre>C:\Z>python ep2_node_to_node.py 6001 6000 Send a message to the peer node and receive message from the peer node. To exit send message: exit. Enter message: hi Message received: hello Enter message: exit Message received: exit</pre>
