

Primer okruženja za pisanje distribuiranih programa na bazi razmene poruka

Ovde je dato jedno okruženje za pisanje distribuiranih programa u obliku API-ja za klijente u čvorovima mreže, i jedan primer distribuiranog programa za ćaskanje (chat). API i primer u ovoj vežbi predstavljaju generalizaciju predhodne vežbe, koju iz tog razloga treba predhodno dobro savladati.

Glavna referenca je Python online dokumentacija:

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

Okruženje za pisanje distribuiranih programa na bazi razmene poruka

Okruženje je definisano u modulu `msg_passing_api.py`, i sastoji se od API-ja i funkcije lokalnog servera.

API funkcija `broadcastMsg` šalje datu poruku svim udaljenim čvorovima u mreži (njeni parametri su lista adresa udaljenih servera i poruka koju treba poslati). Ona koristi API funkciju `sendMsg` (Vežba 3).

API funkcija `rcvMsgs` prima poruke od svih udaljenih čvorova u mreži (njeni parametri su red poruka od lokalnog servera i broj udaljenih čvorova u mreži). Ona koristi API funkcije `rcvMsg` (Vežba 3).

Tabela 1.a: API za pisanje distribuiranih programa (u modulu `msg_passing_api.py`).

```
def broadcastMsg(list_of_remote_server_address, msg):
    for remote_server_address in list_of_remote_server_address:
        sendMsg(remote_server_address, msg)

def rcvMsgs(queue, no_of_messages_to_receive):
    msgs = []

    for i in range(no_of_messages_to_receive):
        msgs.append( rcvMsg(queue) )

    return msgs

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()
```

Funkcija lokalnog servera je identična kao u predhodnoj vežbi. Ukratko: lokalni server prvo postavi svoju adresu i napravi objekt listener, a 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 1.b: Funkcija lokalnog servera (u modulu msg_passing_api.py).

```
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
```

Primer distribuiranog programa za časkanje

Primer komunikacije u potpuno pevezanoj mreži je u modulu example_complete_graph.py. Ovaj program ima dva argumenta komande linije: *proc_index* i *number_of_proc*.

Program je potrebno pokrenuti kao *number_of_proc* različitih instanci (npr. *number_of_proc* = 3), tj. u *number_of_proc* odvojenih terminala, ali sa različitim vrednostima komandnih argumenata. Prvi argument komandne linije treba postaviti na indeks posmatranog čvora (tj. procesora), a drugi na broj čvorova u potpuno povezanoj mreži.

Lokalni klijent – obrada komandne linije i inicijalizacija. Argumenti komandne linije su indeks čvora i broj udaljenih čvorova (svi čvorovi rade na istom računaru 'localhost', a port i-tog čvora je 6000+i). Bitni koraci inicijalizacije su: (1) napravi liste portova, (2) napravi red queue, (3) napravi i pokreni proces lokalnog server, (4) postavi listu adresa udaljenih servera, i (5) ispiši korisničko uputstvo za slanje poruka.

Tabela 1.c: Lokalni klijent – obrada komandne linije i inicijalizacija.

```
def main():
    # Parse command line arguments
    if len(sys.argv) != 3:
        print('Program usage: example_complete_graph proc_index number_of_proc')
        print('Example: If number_of_proc = 3, we must start 3 instances of program in 3 terminals:')
        print('example_complete_graph 0 3, example_complete_graph 1 3, and example_complete_graph 2 3')
        exit()

    # Process command line arguments
    proc_index = int( sys.argv[1] )
```

```

number_of_proc = int( sys.argv[2] )

# Creat list of all pors
allPorts = [6000+i for i in range(number_of_proc)]

# Set ports
local_port = allPorts[proc_index]
remote_ports = [x for x in allPorts if x != local_port]

# 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 lst of the addresses of the peer node's servers
remote_server_addresses = [('localhost', port) for port in remote_ports]

# 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) prekine petlju ako je primio poruku 'exit', (3) pošalje tu poruku udaljenim serverima (pomoću API funkcije broadcastMsg), (4) primi listu poruka od svog lokalnog server (pomoću funkcije rcvMsgs), i (5) ištaampa listu poruka.

Tabela 1.d: Lokalni klijent – beskonačna petlja i završetak.

```

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

    if msg == 'exit':
        sendMsg( ('localhost', local_port), 'exit')
        break

    # Send message to peer node's servers
    broadcastMsg(remote_server_addresses, msg)

    # Get message from local node's server
    msgs = rcvMsgs(queue, number_of_proc - 1)
    print('Messages received:', msgs)

# Join with server process

```

```
server.join()

# Delete queue and server
del queue
del server
```

Primer pokretanja tri instance programa example_complete_graph.py u tri terminala

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

Tabela 1.e: Rezultat izvršenja modula example_complete_graph.py u prvom terminalu.

```
C:\Z>python example_complete_graph.py 0 3
Send a message to the peer node and receive message from the peer node.
To exit send message: exit.
Enter message: terminal1
Messages received: ['terminal2', 'terminal3']
Enter message: exit
```

Tabela 1.f: Rezultat izvršenja modula example_complete_graph.py u drugom terminalu.

```
C:\Z>python example_complete_graph.py 1 3
Send a message to the peer node and receive message from the peer node.
To exit send message: exit.
Enter message: terminal2
Messages received: ['terminal1', 'terminal3']
Enter message: exit
```

```
C:\Z>python example_complete_graph.py 2 3
Send a message to the peer node and receive message from the peer node.
To exit send message: exit.
Enter message: terminal3
Messages received: ['terminal1', 'terminal2']
Enter message: exit
```

Ove instance programa je moguće i automatski pokrenuti pomoću skripte run.bat:

```
@echo off

echo Start 3 instances of example_complete_graph in 3 terminals

start python example_complete_graph.py 0 3
start python example_complete_graph.py 1 3
start python example_complete_graph.py 2 3
```

Zadaci sa samostalni rad

1. U laboratoriji: Pokrenuti četiri instance programa `python example_complete_graph.py` u četiri različita terminala, prvo ručno a zatim sa prerađenom skriptom `run.bat`.
2. Kod kuće: Koristeći dato okruženje (tj. API), napisati distribuirani algoritam plavljenja (vidi objašenje ovog algoritma i sliku 2.4 na str. 14 u knjizi) u potpuno povezanoj mreži. Natuknica: neka je čvor 0 koren mreže, tj. od njega kreće plavljenje. Zadatak treba rešiti preradom prikazanog programa za distribuirano časkanje.