ESTI - Escola Superior da Tecnologia da Informação

EDC - Graduação em Engenharia de Computação

Desenvolvimento Python para Redes e Sistemas Operacionais

TP3

Aluno: Eloy Francisco Barbosa

Professor: Cassius Figueiredo

Data: 16/03/2019

Sumário

1. 3
2. 3
3. 3
4. 3
5. 3
6. 4
7. 4
8. 4
9. 5
10. 6
11. 7

1.
O que é um processo cliente?

Um cliente corresponde ao programa que irá se conectar a um servidor e fazer requisições a ele. É um programa ativo.

2.
O que é um processo servidor?

Um servidor está relacionado ao programa que ficará esperando clientes se conectarem e fazerem requisições a ele. É um programa passivo.

3.
A função socket() do módulo 'socket' de Python é responsável por criar um socket no processo tanto para protocolo TCP, quanto UDP. Como diferenciar se o socket a ser criado é TCP e UDP?

No socket type para tcp usamos SOCK_STREAM e para udp usamos SOCK_DGRAM

Exemplo TCP: socket.socket (socket.AF_INET, socket.SOCK_STREAM)

Exemplo UDP: socket.socket (socket.AF_INET, socket.SOCK_DGRAM)

4.
Para sockets TCP, responda:

a. **Que sequência de chamadas de funções em Python deve ser realizada pelo cliente? (Não precisa especificar os parâmetros)**

   Cria um socket -> conecta um socket local a um socket remoto (pré-especificado por IP ou nome juntamente com a porta de acesso)-> envia/recebe dados -> fecha a conexão.

   socket() -> connect() -> recv() send() -> close()

b. **Que sequência de chamadas de funções em Python deve ser realizada pelo servidor? (Não precisa especificar os parâmetros)**

   Cria um socket -> associa o socket a uma porta e ao endereço local -> permite que o socket criado aceite conexão -> aceita a conexão com o cliente quando requisitado -> envia/recebe dados -> Fecha a conexão.

   socket() -> bind() -> listen() -> accept -> send() recv() -> close()

c. **Quais destas funções são bloqueantes, isto é, o processo fica esperando?**

   connect(), recv() send() ,listen()

5.
Para sockets UDP, responda:

a. **Que sequência de chamadas de funções em Python deve ser realizada pelo cliente? (Não precisa especificar os parâmetros)**

   Cria um socket > associa o socket a um endereço e porta destino > se comunica com o servidor > fecha a conexão.

   socket() -> sendto() recvfrom() -> close()

b. **Que sequência de chamadas de funções em Python deve ser realizada pelo servidor? (Não precisa especificar os parâmetros)**

   Cria um socket > associa o socket a um endereço e porta local > se comunica com o cliente> fecha a conexão.

   socket() -> bind() -> recvfrom() sendto() -> close()

c. **Quais destas funções são bloqueantes, isto é, o processo fica esperando?**

   Nenhuma

6.
Para que serve o comando socket.bind()?

Associa o socket a um endereço e uma porta.

7.
Em sockets Python, como é representado um endereço de um processo remoto?

Pelo Ip e pela Porta.

8.
Crie um programa cliente que:

a. Conecte-se a um servidor via UDP de mesmo IP e porta 9991.

b. Peça ao servidor que envie a quantidade total e disponível de armazenamento do disco principal.

c. Receba e exiba a informação.

```
import socket

print('Nesta atividade vamos nos conectar a um servidor via UDP,
coletar as informações de quantidade total e disponivel\nde
armazenamento do disco na qual o processo do servidor está
rodando.\n')
UDPClient = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
ServerName = socket.gethostname()
Port = 9991

msg = input("Deseja receber as informações(y/n):")

try:
    UDPClient.sendto (msg.encode('utf-8'), (ServerName, Port))

    (msg, client) = UDPClient.recvfrom(1024)
    print(msg.decode('utf-8'))

    input('\nPressione enter para sair...')
    UDPClient.close()

except Exception as error:
    print(error)
```

9.
Associado à questão anterior, crie um programa servidor que:

a. Espere conexões UDP de processos na porta 9991.

b. Aguarde indefinidamente conexão de clientes.

c. Sirva cada cliente com a informação da quantidade total e disponível de armazenamento do disco principal (diretório corrente que o processo servidor está executando).

```
import socket
import psutil

UDPServer = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
ServerName = socket.gethostname()
Port = 9991
UDPServer.bind((ServerName, Port))

principal_total = round(psutil.disk_usage('.').total / (1024 *
1024 * 1024), 2)
```

```
principal_disponivel = round(psutil.disk_usage('c:').free /
(1024 * 1024 * 1024), 2)

try:
    print('Aguardando requisição na porta', Port, '...')
    (msg, client) = UDPServer.recvfrom(1024)
    print('Requisição aceita')
    if msg.decode() == 'y':
        UDPServer.sendto('\nNome do Servidor: {} \nEspaço total:
{}GB \nDisponivel: {}GB'.format(ServerName, principal_total,
principal_disponivel).encode('utf-8'), client)
        print('Informações enviadas com sucesso')

    else:
        UDPServer.sendto('Erro: Digite "y".'.encode('utf-8'),
client)
        print('Informações não foram enviadas.')

except Exception as error:
    print(error)
```

10.
Crie um programa cliente que:

a. Conecte-se a um servidor via TCP de mesmo IP e porta 8881.

b. Envie ao servidor o nome de um arquivo para que ele transmita este arquivo para o cliente.

c. Receba o tamanho do arquivo.

d. Se o tamanho for válido, receba o arquivo. Caso contrário, avise ao usuário que o arquivo não foi encontrado.

```
import socket
import pickle
import os

socket_tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host, port = socket.gethostname(), 8881


def Cliente():
    socket_tcp.connect((host, port))

    file = "arquivo.txt"
    socket_tcp.send(file.encode('utf-8'))

    msg = socket_tcp.recv(8)

    size = int(msg.decode())

    if size >= 0:

        print(f"Tamanho do arquivo {size}")
```

```
        if os.path.isdir('transferidos') is not True:
            os.mkdir('transferidos')

        file = open('transferidos/' + file, 'wb')
        bytes_count = socket_tcp.recv(4096)

        count = 0

        while bytes_count:
            file.write(bytes_count)
            count += len(bytes_count)
            loading(count, size)
            if count == size:
                break
            bytes_count = socket_tcp.recv(4096)
        print("Transferência Concluida!")

    else:
        print("Arquivo não encontrado!")

    socket_tcp.close()


def loading(bytes, size):
    kbytes = bytes / 1024
    tam_bytes = size / 1024
    txt = 'Baixando... '
    txt += '{:<.2f}'.format(kbytes) + ' KB '
    txt += 'de ' + '{:<.2f}'.format(tam_bytes) + ' KB'
    print(txt)

Cliente()
```

11.
Associado à questão anterior, crie um programa servidor que:

a. Espere conexões TCP de processos na porta 8881.

b. Aguarde indefinidamente conexão de clientes.

c. Receba a requisição do arquivo do cliente e envie o seu tamanho, caso o tenha encontrado. Em caso negativo, envie um valor inválido -1.

d. Envie o arquivo para o cliente, caso o encontre.

```
import socket
import os


def Servidor():
    socket_tcp = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    host, port = socket.gethostname(), 8881
```

```
    try:
        socket_tcp.bind((host, port))
        socket_tcp.listen(5)
        print("Servidor de nome %s esperando conexão na porta
%s" % (host, port))

        while True:
            (client, addr) = socket_tcp.accept()
            file = client.recv(1024)
            file = file.decode('utf-8')
            error = -1

            if os.path.isfile(file):

                size = os.stat(file).st_size
                client.send(str(size).encode())

                socket_tcp = open(file, 'rb')
                bytes_count = socket_tcp.read(4096)

                while bytes_count:
                    client.send(bytes_count)
                    bytes_count = socket_tcp.read(4096)

            else:
                client.send(str(error).encode('ascii'))
                print('O arquivo não existe!')

    except Exception as error:
        print(error)
    socket_tcp.close()

Servidor()
```