♦

Questões - 01 (Código)

Questão 01:

> Desenvolver uma aplicação em rede com arquitetura cliente/servidor e protocolo requisição/resposta.

> Cliente envia requisição no formato de uma string com separadores. exemplo: "add,5,3"

> Servidor recebe a requisição, interpreta a mensagem e executa a calculadora. Depois retorna o resultado para o cliente como uma mensagem.

CALCULADORA TCP - CLIENTE

```
from socket import*  # Importa todas as funções necessárias d

# Define o endereço do servidor e a porta que será usada para
serverName = 'localhost'
serverPort = 13000

# Cria um socket do cliente para comunicação usando TCP/IP
clientSocket = socket(AF_INET, SOCK_STREAM)

# Estabelece a conexão com o servidor usando o endereço e por
clientSocket.connect((serverName, serverPort))
```

```
while True:  # Loop infinito para continuar a receber entrada
    print("")  # Imprime uma linha vazia para espaçamento

    # Solicita ao usuário que insira o primeiro número
    num1 = input('Digite o primeiro número: ')

    # Solicita ao usuário que insira o operador
    op = input("Digite o operador: ")

    # Solicita ao usuário que insira o segundo número
    num2 = input('Digite o Segundo número: ')

    if op == "/" and num2 == "0":  # Verifica se a operação é
        print("")
        print("Divisão por zero não é permitida")  # Exibe um
        print("#-----------------------------------------
    else:
        dados = f"{op},{num1},{num2}"  # Formata os dados a s
        clientSocket.send(dados.encode('utf-8'))  # Envia os

        calculo = clientSocket.recv(1024)  # Recebe a respost
        resultado = calculo.decode('utf-8')  # Decodifica a r
        print("")
        print("Resultado:", int(resultado))  # Exibe o result
        print("#-----------------------------------------

    break  # Sai do loop infinito após a primeira iteração
```

CALCULADORA TCP - SERVER

```
import time
from socket import *
serverPort = 13000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(5)
print("O servidor está pronto para receber")
```

```
while 1:
     connectionSocket, addr = serverSocket.accept()

     dados = connectionSocket.recv(1024).decode()
     print(dados)
     op, n1, n2 = dados.split(',')

     calculo = 0

     if op == "+":
          calculo = int(n1)+int(n2)
     elif op == "-":
          calculo = int(n1)-int(n2)
     elif op == "*":
          calculo = int(n1)*int(n2)
     elif op == "/":
          if n2 != 0:
               calculo = int(n1)/int(n2)

     connectionSocket.send(str(calculo).encode('utf-8'))
```

Questão 02:

> Desenvolva um Chat sobre TCP a partir dos códigos baixados na Questão 1. O
> Chat pode ser entre 2 processos apenas. O que deve ser modificado na classe TCPClient?

> Modifique o Chat para uma versão não-bloqueante em que as mensagens podem
> ser enviadas a qualquer tempo depois que uma conexão está estabelecida. Utilize Threads para desbloquear o recebimento de mensagens.

COMUNICAÇÃO

```
from socket import *  # Importa todas as funções necessárias (

class TCPClient:
    def __init__(self, server_name, server_port):
        self.server_name = server_name  # Armazena o nome do
        self.server_port = server_port  # Armazena a porta do
        self.client_socket = socket(AF_INET, SOCK_STREAM)  # (
        self.client_socket.connect((server_name, server_port)

    def send_message(self, message):
        self.client_socket.send(message.encode('utf-8'))  # E

    def receive_message(self):
        modified_sentence = self.client_socket.recv(1024)  # R
        return modified_sentence.decode('utf-8')  # Decodifica

    def close_connection(self):
        self.client_socket.close()  # Fecha a conexão com o s

# Inicialização do cliente, passando o nome do servidor e a p
client = TCPClient('localhost', 12000)

while True:  # Loop infinito para manter a comunicação contín
    # Solicita ao usuário que insira uma mensagem a ser envia
    sentence = input('Mensagem: ')

    # Envia a mensagem para o servidor
    client.send_message(sentence)

    # Recebe e exibe a resposta do servidor
    modified_sentence = client.receive_message()
    print("Servidor:", modified_sentence)
```

```
# Fecha a conexão com o servidor
client.close_connection()
```

COMUNICAÇÃO NÃO-BLOQUEANTE

```
from socket import *
import threading  # Importa o módulo threading para criar thr

class TCPClient:
    def __init__(self, server_name, server_port):
        self.server_name = server_name  # Armazena o nome do
        self.server_port = server_port  # Armazena a porta do
        self.client_socket = socket(AF_INET, SOCK_STREAM)  # 
        self.client_socket.connect((server_name, server_port)
        self.receive_thread = threading.Thread(target=self.re
        self.receive_thread.start()  # Inicia a thread

    def send_message(self, message):
        self.client_socket.send(message.encode('utf-8'))  # E

    def receive_messages(self):
        while True:  # Loop infinito para receber mensagens c
            try:
                modified_sentence = self.client_socket.recv(1
                if not modified_sentence:  # Verifica se a me
                    break  # Se estiver vazia, encerra o loop
                print("\nServidor:", modified_sentence.decode
            except ConnectionAbortedError:  # Captura exceção
                break  # Encerra o loop se a conexão for ence

    def start_chat(self):
        while True:  # Loop infinito para enviar mensagens co
            sentence = input('Mensagem: ')  # Solicita ao usu
            self.send_message(sentence)  # Envia a mensagem p

    def close_connection(self):
```

```
        self.client_socket.close()  # Fecha a conexão com o s

# Inicializa o cliente, passando o nome do servidor e a porta
client = TCPClient('localhost', 12000)

# Inicializa o loop para receber e enviar mensagens
client.start_chat()

# Aguarda até que a thread de recebimento de mensagens termin
client.receive_thread.join()

# Fecha a conexão com o servidor
client.close_connection()
```

Questão 03:

> Adicione um serviço simples de sua escolha ao processo servidor. Quais modificações são necessárias para oferecer dois serviços no mesmo processo? Compare essa solução com a criação de um processo servidor para cada serviço

> Continuando a evolução da arquitetura Cliente-Servidor vista na Lista Prática - 1, devemos retirar a interação com o usuário da classe TCPClient e o serviço da classe TCPServer. Dessa forma, os objetos de TCPClient e TCPServer tornam-se coesos com o único objetivo de prover comunicação (estabelecimento de conexão e trocas de mensagens através dos fluxos de
> entrada (IN) e saída (OUT)).

```
from socket import *
```

```
# Configurações do servidor
serverName = 'localhost'
serverPort = 13000

# Criação do socket do cliente
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))

print("")
print("Qual serviço você deseja acessar?")
print("1 - Calculadora Básica")
print("2 - Fatorial")
print("")

# Solicita ao usuário que escolha um serviço
servico = input("Escolha 1 ou 2: ")

# Se o serviço escolhido for uma calculadora básica
if servico == "1":
    num1 = input('Digite o primeiro número: ')
    op = input("Digite o operador: ")
    num2 = input('Digite o Segundo número: ')

    # Verifica se a divisão por zero foi tentada
    if op == "/" and num2 == "0":
        print("")
        print("Divisão por zero não é permitida")
        print("#---------------------------------------------
    else:
        # Monta a mensagem com os dados da operação
        dados = f"{servico},{op},{num1},{num2}"
        clientSocket.send(dados.encode('utf-8'))

        # Recebe o resultado da operação
        calculo = clientSocket.recv(1024)
        resultado = calculo.decode('utf-8')
        print("")
        print("Resultado:", int(resultado))
```

```
        print("#-----------------------------------------------

# Se o serviço escolhido for o cálculo de fatorial
elif servico == "2":
    número = input("Digite o número para calcular o fatorial:

    # Monta a mensagem com os dados do cálculo
    dados = f"{servico},{número}"
    clientSocket.send(dados.encode('utf-8'))

    # Recebe o resultado do cálculo
    calculo = clientSocket.recv(1024)
    resultado = calculo.decode('utf-8')
    print("")
    print("Resultado:", int(resultado))









# Servidor

# Criação do socket do servidor
serverPort = 13000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(5)
print("O servidor está pronto para receber")

while 1:
     connectionSocket, addr = serverSocket.accept()  # Aceita

     dados = connectionSocket.recv(1024).decode()  # Recebe o
```

```
     tamanho = dados.split(',')
     # Verifica o tipo de serviço solicitado
     if len(tamanho) == 4:
          serviço, op, n1, n2 = dados.split(',')
     else:
          serviço, número = dados.split(',')

     # Se for uma calculadora básica
     if serviço == "1":
          calculo = 0

          # Realiza a operação solicitada
          if op == "+":
               calculo = int(n1)+int(n2)
          elif op == "-":
               calculo = int(n1)-int(n2)
          elif op == "*":
               calculo = int(n1)*int(n2)
          elif op == "/":
               if n2 != 0:
                    calculo = int(n1)/int(n2)

          # Envia o resultado da operação de volta para o cli
          connectionSocket.send(str(calculo).encode('utf-8'))

     # Se for um cálculo de fatorial
     elif serviço == "2":
          if int(número) > 1:
               calculo = 1
               i = 1
               # Realiza o cálculo do fatorial
               while i <= int(número):
                    calculo *= i
                    i += 1
          else:
               calculo = 1
          # Envia o resultado do cálculo de volta para o clie
          connectionSocket.send(str(calculo).encode('utf-8'))
```

```
    connectionSocket.close()  # Fecha a conexão com o client
```

Questão 04:

escreva um código de teste de carga para ambos servidores TCP:

● multithread - múltiplos clientes concorrentes
● singlethread - atende um cliente por vez . A abertura de conexão (accept()) está na mesma thread que trata a requisição (thread main).

- O código de teste deve gerar 100 clientes (threads) simultâneos. Cada cliente fará uma única equisição ao servidor e encerrará. Faça uma rodada para o servidor multithread e outra para o singlethread. Compare os tempos de execução.
- Não faça interação com o usuário no cliente (entrada por teclado). Todas as requisições podem ser iguais e definidas no próprio código: exemplo - “ADD;10;11”.
- Como a calculadora não oferece carga suficiente para a thread perder a CPU antes de terminar sua execução, adicione uma pausa no próprio código. Em java, por exemplo: Thread.sleep(100) (“a thread será suspensa por 100 milissegundos”). Dessa forma, o servidor single thread vai levar pelo menos 100 × 100 milissegundos para tratar todas as 100 requisições.
- Dica: evite mandar mensagem de saída para o console pois ele será uma gargalo para as threads concorrentes (println(), print(), System.out.println()).
- Dica: O tempo de fim só deve ser contabilizado quando todas as threads estiverem encerradas.
- Utilize o join() para sincronizar a thread main do teste de carga com as demais threads que fazem as requisições. - threads[i].join()

Transforme os Objetos que fornecem o serviço (calculadora, por exemplo) em Singletons
● java:
https://www.devmedia.com.br/padrao-de-projeto-singleton-em-java/26392
● python:
https://refactoring.guru/pt-br/design-patterns/singleton/python/example

CLIENTE_ST-MT

```
##----------CLIENTE----------##

import socket  # Importa a biblioteca de sockets
import threading  # Importa a biblioteca para criação de thre
import time  # Importa a biblioteca de tempo

requisicao = "ADD;20;50"  # Define a requisição a ser enviada
clientes = 100  # Define o número de clientes a serem simulad

# Função para o servidor multithreaded
def multiThread_server():
    tempo_inicial = time.time()  # Marca o tempo inicial

    threads = []  # Lista para armazenar as threads
    for i in range(clientes):  # Loop para criar threads de c
        thread = threading.Thread(target=enviar_requisicao_mu
        thread.start()  # Inicia a thread
        threads.append(thread)  # Adiciona a thread à lista

    for thread in threads:  # Loop para aguardar o término de
        thread.join()  # Aguarda o término da thread

    tempo_final = time.time()  # Marca o tempo final
    print("Tempo total para o servidor multithreaded:", tempo_

# Função para o servidor single-threaded
def singleThread_server():
```

```
    tempo_inicial = time.time()  # Marca o tempo inicial

    for _ in range(clientes):  # Loop para simular clientes
        enviar_requisicao_single()  # Envia a requisição ao s

    tempo_final = time.time()  # Marca o tempo final
    print("Tempo total para o servidor single-threaded:", tem

# Função para enviar requisição a um servidor multithreaded
def enviar_requisicao_multi():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
        s.connect(('localhost', 13000))  # Conecta ao servido
        s.sendall(bytes(requisicao, 'utf-8'))  # Envia a requ
        s.recv(1024)  # Aguarda a resposta

# Função para enviar requisição a um servidor single-threaded
def enviar_requisicao_single():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
        s.connect(('localhost', 12000))  # Conecta ao servido
        s.sendall(bytes(requisicao, 'utf-8'))  # Envia a requ
        s.recv(1024)  # Aguarda a resposta

# Inicia uma thread para o servidor multithreaded
threading.Thread(target=multiThread_server).start()
# Inicia o servidor single-threaded
singleThread_server()
```

```
##----------SERVER----------##


import socket  # Importa a biblioteca de sockets
import threading  # Importa a biblioteca para criação de thre

# Função para realizar cálculos com base na requisição recebi
def Calculo(client_socket,  tipo):
    # Recebe a requisição do cliente e a decodifica
    request = client_socket.recv(1024).decode()
```

```
    # Divide a requisição nos operadores e operandos
    operator, n1, n2 = request.split(";")

    # Realiza a operação correspondente e imprime o resultado
    if operator == "ADD":
        print(f"{tipo}:{n1} + {n2} =",int(n1) + int(n2))
    elif operator == "SUB":
        print(f"{n1} - {n2} =",int(n1) - int(n2))
    elif operator == "MUL":
        print(f"{n1} * {n2} =",int(n1) * int(n2))
    elif operator == "DIV":
        # Verifica se a divisão é por zero antes de realizar 
        if int(n2) != 0:
            print(f"{n1} / {n2} =",int(n1) / int(n2))
        else:
            print(f"{tipo}: ERRO Divisão  por zero")

    # Fecha o socket do cliente
    client_socket.close()

# Função para o servidor multithreaded
def multiThread_server():
    # Cria um socket TCP
    server_socket = socket.socket(socket.AF_INET, socket.SOCK
    # Faz o bind do socket ao endereço e porta especificados
    server_socket.bind(('localhost', 13000))
    # Inicia o modo de escuta, permitindo até 5 conexões pend
    server_socket.listen(5)

    print("Servidor multiThreads TCP esperando conexões...")

    while True:
        # Aceita a conexão do cliente
        client_socket, _ = server_socket.accept()
        # Define o tipo de servidor como "Multi Threads"
        tipo = "Multi Threads"
        # Inicia uma nova thread para tratar a conexão do cli
        client_handler = threading.Thread(target=Calculo, arg
```

```
        client_handler.start()

# Função para o servidor single-threaded
def singleThread_server():
    # Cria um socket TCP
    server_socket = socket.socket(socket.AF_INET, socket.SOCK
    # Faz o bind do socket ao endereço e porta especificados
    server_socket.bind(('localhost', 12000))
    # Inicia o modo de escuta, permitindo até 5 conexões pend
    server_socket.listen(5)

    print("Servidor singleThread TCP esperando conexões...")

    while True:
        # Aceita a conexão do cliente
        client_socket, _ = server_socket.accept()
        # Define o tipo de servidor como "Single Thread"
        tipo = "Single Thread"
        # Chama a função de cálculo para tratar a conexão do
        Calculo(client_socket, tipo)

# Inicia uma nova thread para o servidor multithreaded
threading.Thread(target= multiThread_server).start()
# Inicia o servidor single-threaded
singleThread_server()
```