UNIVERSIDADE FEDERAL DO CEARÁ

LISTA 1

Aluna: Larissa da Silva Matos
Matrícula: 494223

PROJETO DISPONÍVEL EM:
https://github.com/Lrs-mtos/sistemas-distribuidos/tree/master/tcp_python

Questão 1 – 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.

Resposta:
A resposta na questão 2 mostra como isso poderia ocorrer, porém, a diferença no caso de um serviço rodando no mesmo código do processo servidor é que não haveria um arquivo chamado *temp_converter*, pois as funções nesse arquivo seriam métodos de uma classe e todas as ações de conversão deveriam ser feitas da seguinte forma dentro de server_tcp.py:

```
if operation == "c2k":
        result == temp_converter.celsius2kelvin(*operands)
    elif operation == "k2c":
        result = temp_converter.kelvin2celsius(*operands)
    elif operation == "k2f":
        result = temp_converter.kelvin2fahrenheit(*operands)
    elif operation == "f2k":
        result = temp_converter.fahrenheit2kelvin(*operands)
    elif operation == "c2f":
        result = temp_converter.celcius2fahrenheit(*operands)
```

Questão 2 - 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))

Resposta:
O serviço adicionado foi um conversor de temperaturas:

- Kelvin para Fahrenheit e vice-versa;
- Celsius para Fahrenheit e vice-versa;
- Celsius para Kelvin e vice-versa;

Para que isso fosse possível, foi necessário:

1. Criar um arquivo chamado converter.py com as seguintes linhas de código:

tcp_python > temp_converter.py > ...

You, 4 days ago | 1 author (You)

```python
def celsius2kelvin(c):
    return c + 273

def kelvin2celsius(k):
    return k - 273

def kelvin2fahrenheit(k):
    return (k - 273) * 9/5 + 32

def fahrenheit2kelvin(f):
    return (f - 32) * 5/9 + 273

def celcius2fahrenheit(c):
    return c * 9/5 + 32
    You, 4 days ago • temperature conversion service added
def fahrenheit2celcius(f):
    return (f - 32) * 5/9
```

2. Importar esse arquivo dentro do arquivo que contém o serviço de conexão com o servidor com import *temp_converter*

3. Criar um dicionário chamado *temperatures* (opcional) para organizar e determinar as palavras-chave que serão relacionadas às funções específicas de conversão de temperatura, como mostrado a seguir:

tcp_python > server_tcp.py > ...

```python
temperatures = {
    "c2k": temp_converter.celsius2kelvin,
    "k2c": temp_converter.kelvin2celsius,
    "k2f": temp_converter.kelvin2fahrenheit,
    "f2k": temp_converter.fahrenheit2kelvin,
    "c2f": temp_converter.celcius2fahrenheit,
    "f2c": temp_converter.fahrenheit2celcius,
}
```

4. Receber os dados do usuário e verificar se estes estão de acordo com os serviços providenciados pelo servidor. A função *perform_operation* será mostrada no passo 5.

```python
parts = data.split()
operation = parts[0]
operands = [int(operand) for operand in parts[1:]]

response = perform_operation(operation, operands, operations) or perform_operation(operation, operands, temperatures)
if response is None:
    raise ValueError("Operation not supported by this server".encode("utf-8"))
```

5. Criar uma função que recebe a operação (c2k, k2c, etc.), operando(s) (valor da temperatura) a função específica (dicionário) e que realiza a operação a partir desses parâmetros. A maior parte da checagem de serviços é feita nessa função, que retorna o resultado se tudo ocorrer bem, mas retorna None, caso contrário.

```python
def perform_operation(operation, operands, functions):
    if operation in functions:
        result = functions[operation](*operands)
        return str(result).encode("utf-8")
    else:
        return None
```

6. A parte do cliente ficou separada em duas etapas: estabelecer a conexão com o servidor e receber operandos do usuário, respectivamente:

tcp_python > client_tcp.py > ...

You, 6 minutes ago | 1 author (You)

```python
import socket
import operands

HOST = "127.0.0.1"  # The server's hostname or IP address
PORT = 1024  # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))

    while True:

        operation = operands.get_operands()
        if operation is None:
            s.close()
            break

        s.sendall(operation.encode("utf-8"))
        result = s.recv(1024)
        print(f"Result: {result.decode('utf-8')}")
```

You, last week • codigos de socket …

tcp_python > operands.py > ...

```python
...
def get_operands():
    """Get operands/temperature from user input."""
    operation = input("Enter operation/temperature or 'exit' to quit: ")
    if operation == "exit":
        return (None)
    return operation
```

Adicionar serviços a um processo servidor já existente, pode deixar o código mais complexo, pois passa a ser necessário gerenciar a lógica de ambos os serviços em um único código. Enquanto que, ao adicionar novas funcionalidades numa abordagem de serviços independentes, cria-se uma padronização para adicionar as novas funcionalidades, deixando que o código do servidor exerça sua função sem ter que tratar operações que não têm relação com a conexão em si, fator pode deixar o sistema com um baixíssimo desempenho dependendo da complexidade.

Além disso, existem problemas ainda mais graves que podem surgir: em caso de concorrência e tolerância a falhas, ter um processo separado para cada serviço facilita a implementação de estratégias de recuperação de falhas específicas para cada serviço. Ademais, se uns serviços forem mais custosos que outros, esses podem acabar sendo negativamente afetados.

A escalabilidade horizontal é mais simples quando se temos vários serviços a serem oferecidos, porém, se há um número pequeno e limitado, é possível que seja vantajoso deixá-los rodando com o servidor.

Aliás, ainda existem questões de segurança, comunicação e testabilidade que podem ser influenciadas de maneira ruim caso todos os serviços estejam concentrados em um único lugar.