UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N
ATIVIDADE DE PROGRAMAÇÃO GUIADA: REMOTE PROCEDURE CALLS

OBJETIVO DA AULA PRÁTICA:

Esta atividade de Programação Guiada tem por objetivo demonstrar o funcionamento de Chamadas de Procedimento Remota (Remote Procedure Calls). Para isso, uma aplicação simples que realiza a troca de mensagens através do protocolo gRPC (Google RPC) deverá ser desenvolvida. Nesta aula será utilizado o framework gRPC para linguagem Python 3.

Tutorial de apoio:
- Full pytest documentation
- Quick start | Python | gRPC

INSTALAÇÃO E CONFIGURAÇÃO DE DEPENDÊNCIAS:

Para essa atividade de programação guiada, vamos precisar dos softwares abaixo.

- Python 3. Para instruções de instalação, acesse https://docs.python.org/3/using/index.html. No Ubuntu Linux, você pode instalar o Python 3 com o seguinte comando:

```
apt-get install python3 python3-pip
```

Com o comando acima, o Python 3 e as principais dependências serão instaladas e configuradas automaticamente.

- Biblioteca `gRPC` e ferramental relacionado

```
python3 -m pip install grpcio
python3 -m pip install grpcio-tools
python3 -m pip install pytest
```

Caso você tenha algum problema nas instalações acima, experimente executar os comandos abaixo:

```
pip3 install --upgrade pip
python3 -m pip install --upgrade setuptools
```

CALCULADORA COMO REMOTE PROCEDURE CALL:

Você deverá implementar um serviço RPC em Python, onde sua função será disponibilizar uma API remota que funcione como uma calculadora. A aplicação cliente irá utilizar o endpoint do serviço para realizar chamadas a sua função.

Passos:

1. **Faça o download da implementação base:** baixe da página do moodle o arquivo **PG2-RPC.zip**, e descompacte-o. Verifique no diretório descompactado quais arquivos estão presentes e examine o conteúdo delas.
2. **Analise o arquivo `calculator.proto`**: serviços gRPC utilizam uma linguagem de definição de interfaces (*Interface Definition Language* - IDL) para descrever os métodos e as mensagens

envolvidas nos procedimentos remotos. Esse arquivo descreve a interface da nossa calculadora. Inicialmente, o arquivo contém o serviço *Calculator* com o método *Sum* e as mensagens *SumRequest* e *SumReply*.

- *SumRequest* é a mensagem de requisição do método *Sum* e que contém dois atributos a e b, ambos do tipo **double**, e que correspondem aos números que serão somados. Os números que são atribuídos a esses atributos indicam a posição dos atributos na mensagem (1 e 2, primeiro e segundo, respectivamente).
- *SumReply* é a mensagem de resposta do método *Sum* e que contém um atributo s do tipo **double**, e que corresponde ao retorno do método. Da mesma forma, atributos em uma mensagem de resposta possuem um número atribuído, que corresponde à posição do atributo na mensagem.

3. **Analise o arquivo `calculator_server.py`:** note que ele inclui uma única classe que implementa a interface `CalculatorServicer`, correspondente ao serviço `Calculator` definido na IDL, e um único método, `Sum`, que realiza a soma de seus operadores. O método `Sum` recebe dois parâmetros, uma `SumRequest`, que encapsula os valores dos operadores do método, e um `ServicerContext`, utilizado para manipular o estado da conexão.
4. **Analise o arquivo `calculator_integration_test.py`**: esse arquivo implementa testes de integração para o serviço de calculadora. A função `grpc_server` mostra como rodar um servidor gRPC e registrar as interfaces da nossa implementação no arquivo `calculator.py`. Notem que o servidor escuta numa porta insegura. A função `channel` mostra como iniciar uma conexão insegura com o servidor gRPC. A função `calculator_client` mostra como instanciar um cliente gRPC para a nossa interface de calculadora. Por último, a função `test_sum` apresenta um teste de integração para o método `Sum` do serviço `Calculator`.
5. **Execute os testes:** testes são partes integrais do ciclo de vida de desenvolvimento de software (*Software Development Life Cycle* - SDLC) moderno. Inicialmente, **os testes falharão** com um `ModuleNotFoundError` pois ainda não geramos as interfaces e mensagens bases da nossa implementação.

```
python3 -m pytest
```

6. **Gere as interfaces e as mensagens gRPC:** nessa etapa, usaremos o ferramental do gRPC para Python3 para gerar as classes bases para nossa implementação. Para tal, utilizamos o seguinte comando.

```
python3 -m grpc_tools.protoc \
    -I. --python_out=. --grpc_python_out=. calculator.proto
```

7. **Analise os arquivos gerados**: o passo anterior gerará dois arquivos, `calculator_pb2.py` e `calculator_pb2_grpc.py`, contendo os descritores das mensagens e dos serviços, respectivamente. Através desses arquivos, desenvolvedores de serviços gRPC podem concentrar-se na implementação dos seus serviços sem se preocuparem com os detalhes do protocolo gRPC.
8. **Re-execute os testes:** com os arquivos gRPC gerados, podemos ver os testes rodando com sucesso através do mesmo comando descrito em um passo anterior. Dessa vez, espera-se que os testes passem com sucesso.

```
python3 -m pytest
```

9. **Descreva o procedimento e as mensagens para multiplicação de valores:** para tal, precisamos "aumentar" nosso serviço de calculadora (`calculator.proto`) com o procedimento `Multiply` e suas mensagens `MultiplyRequest` e `MultiplyReply`. Em seguida, devemos re-gerar as interfaces e as mensagens através do ferramental gRPC.
10. **Implemente o procedimento no arquivo `calculator.py`:** seguindo o exemplo do método `Sum`, desenvolva um método análogo que realize a multiplicação dos valores de entrada.

- Não esqueça de importar a definição das novas mensagens

11. **Implemente os testes de integração no arquivo `calculator_integration_test.py`:** para validar nossa implementação, vamos escrever uma função de teste para multiplicação. No arquivo de testes, crie uma função `test_multiply` análoga a função `test_sum`. Em seguida, re-execute os testes de integração.
12. **Agora, execute os passos 9, 10 e 11 para um método que retorna o maior valor entre três números.** Observe nesse caso que a mensagem de requisição deverá conter três atributos ao invés de dois, como ocorre nos métodos anteriores.
13. **Por fim, execute os passos 9, 10 e 11 para um método que retorna o quociente e o resto de da divisão de dois números.** Observe nesse caso que a mensagem de requisição deverá conter dois atributos (dividendo e divisor) e que a mensagem de resposta também deverá conter dois números (quociente e resto).
    - A mensagem de resposta pode ser construída separando por vírgula as instruções que definem os valores que devem ser definidos para cada atributo da mensagem `request`
    - Você pode forçar um quociente de divisão inteiro convertendo o resultado da divisão para o tipo `int()`
14. **Problemas?** Revise os passos anteriores e corrija!

Yay! Se você chegou até aqui, well done! Faça o envio do relatório do programa no Moodle, e seu objetivo na atividade de programação guiada estará cumprido. O relatório deverá incluir o código fonte desenvolvido e screenshots da execução dos métodos de teste.