Computação Distribuída com RPC (gRPC)

Prof Alcides/Prof Mario

Laboratório (gRPC + C++)

Objetivos de Aprendizagem

1. Definir um serviço gRPC usando Protocol Buffers (Protobuf). (0%)
2. Implementar e executar um servidor e um cliente gRPC em C++. (30%)
3. Implementar uma chamada RPC com streaming de dados (Server Streaming). (30%)
4. Executar o servidor/cliente em Python e C++ de forma intercalada (40%)

Passo 1. O Contrato – notas.proto

Este é o coração do projeto. O arquivo .proto será a fonte da verdade para ambas as implementações:

```
syntax = "proto3";

package gerencia_notas;

// Estrutura para uma nota individual
message Nota {
    string ra = 1;
    string cod_disciplina = 2;
    int32 ano = 3;
    int32 semestre = 4;
    float nota = 5;
}

// Resposta para operações de consulta de uma única nota
message ConsultaNotaResponse {
    bool sucesso = 1;
    Nota nota = 2;
    string msg_erro = 3;
}

// Requisição para adicionar ou consultar uma nota
message AlunoDisciplinaRequest {
    string ra = 1;
    string cod_disciplina = 2;
}

// Requisição para adicionar uma nota completa
message AdicionaNotaRequest {
    string ra = 1;
    string cod_disciplina = 2;
    int32 ano = 3;
    int32 semestre = 4;
    float nota = 5;
}

// Resposta genérica de status (sucesso/falha)
message StatusResponse {
```

```
    bool sucesso = 1;
    string msg = 2;
}

// Requisição para calcular média, identificando apenas o aluno
message AlunoRequest {
    string ra = 1;
}

// Resposta com o valor da média
message MediaResponse {
    bool sucesso = 1;
    float media = 2;
    string msg_erro = 3;
}

// Definição do serviço
service GerenciadorNotas {
    // RPCs Unários (requisição/resposta simples)
    rpc AdicionarNota(AdicionaNotaRequest) returns (StatusResponse) {}
    rpc AlterarNota(AdicionaNotaRequest) returns (StatusResponse) {}
    rpc ConsultarNota(AlunoDisciplinaRequest) returns (ConsultaNotaResponse) {}
    rpc CalcularMedia(AlunoRequest) returns (MediaResponse) {}

    // ***** RPC Desafio *****
    // Server Streaming: cliente pede as notas de um aluno, servidor envia um stream
de notas
    rpc ListarNotasAluno(AlunoRequest) returns (stream Nota) {}
}
```

Parte 2. Implementação em C++

- Por que C++? É a linguagem padrão para sistemas de alta performance. Vamos aprender um pouco sobre gerenciamento de dependências e sistemas de build (CMake), habilidades essenciais em engenharia de software.

2.1. Preparação do Ambiente C++

```
:~$ sudo apt update

:~$ sudo apt install -y build-essential autoconf libtool pkg-config
cmake libgflags-dev libgtest-dev clang libc++-dev libprotobuf-dev
protobuf-compiler libgrpc-dev libgrpc++-dev protobuf-compiler-grpc
```

2.2. Estrutura do Projeto e Build com CMake

Vamos utilizar esse arquivo CMakeLists.txt. Isso abstrai a complexidade da compilação e linkagem.

```
cmake_minimum_required(VERSION 3.10)
project(grpc_lab CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++17")

# Encontra os pacotes de biblioteca
find_package(Protobuf REQUIRED)
find_package(gRPC REQUIRED)

# 1. Encontra os executáveis de compilação de forma explícita
#    Encontra o 'protoc'
find_program(PROTOC_EXECUTABLE protoc)
if(NOT PROTOC_EXECUTABLE)
```

```
    message(FATAL_ERROR "Compilador 'protoc' não encontrado. Verifique se o pacote
'protobuf-compiler' está instalado.")
endif()

#    Encontra o plugin do gRPC para C++ pelo nome correto no Ubuntu
find_program(GRPC_CPP_PLUGIN grpc_cpp_plugin)
if(NOT GRPC_CPP_PLUGIN)
    message(FATAL_ERROR "Plugin 'grpc_cpp_plugin' não encontrado. Verifique se o
pacote 'libgrpc-dev' ou 'protobuf-compiler-grpc' está instalado.")
endif()

# 2. Define os nomes dos arquivos que serão gerados a partir de notas.proto
set(PROTO_SRC_FILES ${CMAKE_CURRENT_BINARY_DIR}/notas.pb.cc)
set(PROTO_HDR_FILES ${CMAKE_CURRENT_BINARY_DIR}/notas.pb.h)
set(GRPC_SRC_FILES ${CMAKE_CURRENT_BINARY_DIR}/notas.grpc.pb.cc)
set(GRPC_HDR_FILES ${CMAKE_CURRENT_BINARY_DIR}/notas.grpc.pb.h)

# 3. Define o comando customizado para gerar os arquivos
add_custom_command(
  OUTPUT ${PROTO_SRC_FILES} ${PROTO_HDR_FILES} ${GRPC_SRC_FILES} ${GRPC_HDR_FILES}
  # Comando para gerar os arquivos .pb.cc/.h (mensagens) usando a variável que
encontramos
  COMMAND ${PROTOC_EXECUTABLE}
    --cpp_out=${CMAKE_CURRENT_BINARY_DIR}
    -I${CMAKE_CURRENT_SOURCE_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/notas.proto
  # Comando para gerar os arquivos .grpc.pb.cc/.h (serviços) usando o plugin que
encontramos
  COMMAND ${PROTOC_EXECUTABLE}
    --grpc_out=${CMAKE_CURRENT_BINARY_DIR}
    --plugin=protoc-gen-grpc=${GRPC_CPP_PLUGIN} # A sintaxe name=path ainda é a mais
correta
    -I${CMAKE_CURRENT_SOURCE_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/notas.proto
  DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/notas.proto
  COMMENT "Gerando código a partir de notas.proto"
)

# 4. Cria um alvo para garantir que a geração de código aconteça
add_custom_target(generate_grpc_files ALL DEPENDS
  ${PROTO_SRC_FILES} ${PROTO_HDR_FILES} ${GRPC_SRC_FILES} ${GRPC_HDR_FILES}
)

# --- Seção de Compilação dos Executáveis ---
# Adiciona o diretório dos cabeçalhos gerados ao include path
include_directories(${CMAKE_CURRENT_BINARY_DIR})

# Adiciona o executável do servidor
add_executable(servidor_cpp servidor.cpp ${PROTO_SRC_FILES} ${GRPC_SRC_FILES})
target_link_libraries(servidor_cpp PRIVATE protobuf::libprotobuf gRPC::grpc++)

# Adiciona o executável do cliente
add_executable(cliente_cpp cliente.cpp ${PROTO_SRC_FILES} ${GRPC_SRC_FILES})
target_link_libraries(cliente_cpp PRIVATE protobuf::libprotobuf gRPC::grpc++)
```

2.3. Implementando o Servidor (servidor.cpp)

```
#include <iostream>
#include <memory>
#include <string>
#include <vector>
#include <map>
#include <numeric>
```

```cpp
#include <grpcpp/grpcpp.h>
#include "notas.grpc.pb.h"

// Usando um std::map para simular o banco de dados em memória
std::map<std::string, gerencia_notas::Nota> db_notas_cpp;

class GerenciadorNotasImpl final : public gerencia_notas::GerenciadorNotas::Service {
public:
    grpc::Status AdicionarNota(grpc::ServerContext* context,
                               const gerencia_notas::AdicionaNotaRequest* request,
                               gerencia_notas::StatusResponse* response) override {
        std::string chave = request->ra() + "_" + request->cod_disciplina();
        if (db_notas_cpp.count(chave)) {
            response->set_sucesso(false);
            response->set_msg("Nota já existe. Use 'AlterarNota'.");
        } else {
            gerencia_notas::Nota nova_nota;
            nova_nota.set_ra(request->ra());
            nova_nota.set_cod_disciplina(request->cod_disciplina());
            nova_nota.set_ano(request->ano());
            nova_nota.set_semestre(request->semestre());
            nova_nota.set_nota(request->nota());
            db_notas_cpp[chave] = nova_nota;
            response->set_sucesso(true);
            response->set_msg("Nota adicionada com sucesso!");
        }
        return grpc::Status::OK;
    }

    grpc::Status AlterarNota(grpc::ServerContext* context,
                             const gerencia_notas::AdicionaNotaRequest* request,
                             gerencia_notas::StatusResponse* response) override {
        //****************************************
        //Implemente o código desta função
    }

    grpc::Status ConsultarNota(grpc::ServerContext* context,
                               const gerencia_notas::AlunoDisciplinaRequest* request,
                               gerencia_notas::ConsultaNotaResponse* response)
override {
        //****************************************
        //Implemente o código desta função
    }

    grpc::Status CalcularMedia(grpc::ServerContext* context,
                               const gerencia_notas::AlunoRequest* request,
                               gerencia_notas::MediaResponse* response) override {
        //****************************************
        //Implemente o código desta função
    }

    // --- Implementação do Desafio ---
    grpc::Status ListarNotasAluno(grpc::ServerContext* context,
                                  const gerencia_notas::AlunoRequest* request,
                                  grpc::ServerWriter<gerencia_notas::Nota>* writer)
override {
        //****************************************
        //Implemente o código desta função
    }
};

void RunServer() {
    std::string server_address("0.0.0.0:50052");
    GerenciadorNotasImpl service;
    grpc::ServerBuilder builder;
    builder.AddListeningPort(server_address, grpc::InsecureServerCredentials());
```

```
    builder.RegisterService(&service);
    std::unique_ptr<grpc::Server> server(builder.BuildAndStart());
    std::cout << "Servidor C++ escutando em " << server_address << std::endl;
    server->Wait();
}

int main(int argc, char** argv) {
    RunServer();
    return 0;
}
```

2.4. Implementando o Cliente (cliente.cpp)

```
#include <iostream>
#include <memory>
#include <string>

#include <grpcpp/grpcpp.h>
#include "notas.grpc.pb.h"

class NotasClient {
public:
    NotasClient(std::shared_ptr<grpc::Channel> channel)
        : stub_(gerencia_notas::GerenciadorNotas::NewStub(channel)) {}

    void testarTudo() {
        //Adicionar Notas
        std::cout << "\n1. Adicionando notas..." << std::endl;
        adicionarNota("789", "FIS0001", 2025, 1, 10.0);
        adicionarNota("789", "MAT0001", 2025, 1, 9.0);
        adicionarNota("101", "FIS0001", 2025, 1, 6.5);

        std::cout << "\n2. Consultando nota de FIS0001 para RA 789..." << std::endl;
        //*****************************************
        //Utilize a função consultarNota aqui

        std::cout << "\n3. Alterando nota de MAT0001 para RA 789 para 9.5..." <<
std::endl;
        //*****************************************
        //Utilize a função alterarNota aqui

        std::cout << "\n4. Calculando média para RA 789..." << std::endl;
        //*****************************************
        //Utilize a função calcularMedia aqui

        std::cout << "\n5. DESAFIO: Listando todas as notas do RA 789 via
streaming..." << std::endl;
        //*****************************************
        //Utilize a função listarNotasAluno aqui
    }

private:
    void adicionarNota(const std::string& ra, const std::string& cod, int ano, int
semestre, float nota_val) {
        gerencia_notas::AdicionaNotaRequest request;
        request.set_ra(ra);
        request.set_cod_disciplina(cod);
        request.set_ano(ano);
        request.set_semestre(semestre);
        request.set_nota(nota_val);

        gerencia_notas::StatusResponse reply;
        grpc::ClientContext context;
        grpc::Status status = stub_->AdicionarNota(&context, request, &reply);
```

```
        if (status.ok()) {
            std::cout << "  - Resposta: " << reply.msg() << std::endl;
        } else {
            std::cout << "  - Erro RPC: " << status.error_message() << std::endl;
        }
    }

    void alterarNota(const std::string& ra, const std::string& cod, int ano, int
semestre, float nota_val) {
        //*****************************************
        //Implemente o código desta função
    }

    void consultarNota(const std::string& ra, const std::string& cod) {
        //*****************************************
        //Implemente o código desta função
    }

    void calcularMedia(const std::string& ra) {
        //*****************************************
        //Implemente o código desta função
    }

    void listarNotasAluno(const std::string& ra) {
        //*****************************************
        //Implemente o código desta função
    }

    std::unique_ptr<gerencia_notas::GerenciadorNotas::Stub> stub_;
};

int main(int argc, char** argv) {
    std::string target_str = "localhost:50052";
    if (argc > 1) {
        target_str = argv[1];
    }
    std::cout << "--- Cliente C++ conectando em " << target_str << " ---" <<
std::endl;
    NotasClient client(grpc::CreateChannel(target_str,
grpc::InsecureChannelCredentials()));
    client.testarTudo();
    return 0;
}
```

Passe 3. Execução

```
:~$ #Workflow para os alunos:

:~$ mkdir build && cd build

:~$ #(Gera os arquivos do .proto e o Makefile)

:~$ cmake ..

:~$ #(Compila o servidor e o cliente)

:~$ make

:~$ #Executar o servidor em um terminal e o cliente em outro

:~$ #terminal 1

:~$ ./servidor
```

```
:~$ #terminal 2
:~$ ./cliente
```

Passo 4. Entrega

1. O exercício deve ser feito em grupo (4 alunos) e deve ser entregue no laboratório rodando, execute primeiro o servidor e depois o cliente, tudo na mesma máquina, execute o cliente e o servidor em C++.
2. Execute o servidor em C++ e o cliente em Python, verifique as portas.
3. Execute o servidor em Python e o cliente em C++, verifique as portas.

OBS: caso queira rodar no codespaces, utilize o comando abaixo se der erro pelas vias normais.

```
:~$ # forçar a libstdc++ do sistema
:~$ # terminal 1
:~$ env LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu ./servidor_cpp
:~$ # terminal 2
:~$ env LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu ./cliente_cpp
```