Universidade Federal do Espírito Santo
Departamento de Informática

André Barreto
Igor Ventorim

Implementação de comunicação entre servidor de arquivos e cliente

Trabalho de Interfaces e Periféricos

Vitória
2016

1 Introdução

Neste trabalho temos como finalidade realizar a comunicação de um servidor de arquivos com um cliente através de um protocolo a ser criado especificamente para tal objetivo.

A solução adotada em nosso trabalho foi a API protocol buffers da google, o protocolo é um mecanismo eficiente, flexível e automatizado adotado no mercado, capaz de realizar a serialização de dados estruturados. O protocolo funciona da seguinte forma, é definido como será a estrutura dos dados, através desta nomeclatura é estruturada a linguagem de marcação gerando um arquivo .proto, cada mensagem do protocolo buffer é um registro lógico pequeno de informações, contendo uma série de pares nome-valor, a partir do qual se gera o arquivo final a ser utilizado na linguagem através de um script disponibilizado pela google, ao qual é capaz de interpretar a estrutura definida no .proto e gerar o código fonte.

A partir do codigo fonte na linguagem específica e seguindo a documentação da API é possível realizar a comunicação através do protocolo definido.

2 Protocolo

O método de comunicação utilizado neste trabalho foi implementado através da API *protocol buffers* do Google. Esta API resolve os aspectos mais tecnológicos da comunicação nos termos das mensagens que devem ser enviadas e como devem ser lidas. Com isto, o problema se resumiu em como serão as mensagens trocadas entre o cliente e servidor, fazendo uma simples configuração em um arquivo *.proto*.

Em suma, deve-se configurar os tipos de mensagens que serão enviados e recebidos pelos dois lados da comunicação. Nesta solução, exitem dois tipos de mensagem: *Command* e *Response*, enviados pelo cliente e servidor, respectivamente. O servidor recebe um Command do cliente e o cliente recebe um Response do servidor para cada comunicação bem sucedida. Para auxiliar na configuração destes dois tipos de mensagem, existem dois enumerados: *CommandType* e *StatusCode*, que resumem os tipos de comandos que o cliente pode solicitar e os tipos de erros que podem ocorrer na execução destas solicitações na parte do servidor, respectivamente.

A seguir, são descritos os tipos enumerados e as mensagens do protocolo.

2.1 Tipos enumerados

CommandType e *StatusCode* são tipos enumerados criados para definir os possíveis casos de solicitação e resultados de execução, utilizados pelo cliente e servidor, respectivamente. Eles são definidos como a seguir:

CommandType

LS	RM
CD	CP
MV	CAT
MKDIR	UPLOAD
RMDIR	DOWNLOAD

Tabela 1: Possíveis atribuições de CommandType

Descrição dos possíveis valores do CommandType:

- LS - Solicita a listagem de arquivos do diretório atual
- CD - Solicita uma mudança de diretório
- MV - Solicita a mudança de um arquivo de local
- MKDIR - Solicita a criação de uma pasta
- RMDIR - Solicita a remoção de uma pasta
- RM - Solicita a remoção de um arquivo
- CP - Solicita a cópia de um arquivo
- CAT - Solicita o envio do conteúdo de um arquivo para mostrar na tela do cliente
- UPLOAD - Solicita o envio ao servidor um arquivo
- DOWNLOAD - Solicita o envio do servidor ao cliente de um arquivo

StatusCode

SUCCESS	ERR_ARGS
ERR_PERMISS	ERR_DESC

Tabela 2: Possíveis atribuições de StatusCode

Descrição dos possíveis valores do StatusCode:

- SUCCESS - Informa ao cliente que a solicitação foi realizada com sucesso. Todos os outros casos são casos de erro.
- ERR_PERMISS - Informa ao cliente que houve erro na solicitação por falta de privilégios de usuário.
- ERR_ARGS - Informa ao cliente que houve erro por conta de algum erro nos argumentos passados junto ao comando desejado
- ERR_DESC - Informa ao cliente que houve algum erro não previsto no âmbito do servidor. Majoritariamente por conta de falhas de baixo nível ao tentar executar uma solicitação que deveria ser bem sucedida.

2.2 Mensagem Command

A mensagem *Command* é a mensagem enviada pelo cliente e recebida pelo servidor. Nela conterão as informações que definirão qual comando está sendo solicitado e todos os argumentos necessários para realizar tal comando. Exceto o primeiro, todos os outros campos são opcionais.

CommandType *type*	string *arg0*	string *arg1*	bytes *data*

Tabela 3: Formato do pacote Command

2.3 Mensagem Response

A mensagem *Response* é a mensagem de resposta do servidor a um comando solicitado pelo cliente. Nesta mensagem são informados o resultado da execução da solicitação, isto é, se foi realizado com sucesso ou qual erro ocorreu, e opcionalmente é enviado dados solicitados pelo cliente. O campo *response* é opcional, utilizado quando é necessário enviar dados ao cliente.

StatusCode *status*	bytes *response*

Tabela 4: Formato do pacote Response

2.4 Processamento das mensagens

Com os tipos de mensagens especificados, existe a lógica básica que o cliente e servidor deverão realizar para entender e utilizar as informações de cada mensagem.

Servidor:

1. Recebe uma solicitação (mensagem do tipo *Command*)

2. Reconhece o *CommandType* da mensagem
3. Recupera os argumentos da mensagem, caso sejam necessários para o comando em questão
4. Realiza a solicitação

Cliente:

1. Recebe uma resposta (mensagem do tipo *Response*)
2. Avalia o *StatusCode* da mensagem, reconhecendo se houve erro ou não na solicitação
3. Realiza processamentos internos de acordo com o *StatusCode*, informando ao usuário o resultado

2.5 Arquivo de configuração do protocolo

Com as especificações descritas nessa seção, segue aqui um exemplo de arquivo de configuração do protocol buffers.

```
syntax = "proto3";

option java_outer_classname = "CommandsProto";

enum CommandType {
  LS = 0;
  CD = 1;
  MV = 2;
  MKDIR = 3;
  RMDIR = 4;
  RM = 5;
  CP = 6;
  CAT = 7;
  UPLOAD = 8;
  DOWNLOAD = 9;
}

enum StatusCode {
  SUCCESS = 0;
  ERR_PERMISS = 1;
  ERR_ARGS = 2;
  ERR_DESC = 3;
```

```
}

message Command {
  CommandType type = 1;
  string arg0 =  2;
  string arg1 = 3;
  bytes data = 4;
}

message Response {
  StatusCode statusCode = 1;
  bytes response = 2;
}
```

3 Implementação

Nossa implementação para uma melhor organização foi dividida em quatro módulos, sendo eles cliente, servidor, protocolo e utilitários.

Cliente: neste módulo foram implementadas todas as funcionalidades que se referem somente ao cliente, sendo composto pela seguintes classes: Client, ClientMain, ClientException e ClientOperation.

- Client: Classe responsável por realizar a comunicação entre a interface e as operações do cliente.
- ClientMain: Classe principal do cliente, responsável pela execução.
- ClientException: Classe que trata as exceções geradas pelo cliente.
- ClientOperarion: Classe de operações do cliente, responsável por tratar todas as funções correspondentes ao domínio do problema.

Servidor: neste módulo foram implementadas todas as funcionalidades que se referem somente ao servidor, sendo composto pelas seguintes classes: Server, ServerMain, ServerException e ServerOperation.

- Server: Classe responsável por realizar a comunicação entre interface e operações do servidor que implementa runnable para que seja possível a conexão de vários clientes no servidor.
- ServerMain: Classe principal do servidor, responsável por sua execução.
- ServerException: Classe responsável por tratar as exceções que ocorrerem no servidor.

- ServerOperation: Classe que contém todas as operações referentes ao domínio do problema no lado do servidor.

Protocolo: este módulo foi separado para o arquivo gerado pelo protocol buffers, contendo a classe CommandsProto ao qual contém todas as funcionalidades responsável por prover a utilização do protocol buffers, onde todo código utilizado é segundo a documentação da google uma caixa preta, ao qual o desenvolvedor só precisa saber os métodos que existem e oque retornam, porém não interessando como os quais funcionam por dentro.

Utilitários: neste módulo foram implementadas todas as funcionalidades que se referem para ambos(cliente e servidor), ao qual contém as classes: CommandCode, Message e StatusCode.

4 Interoperabilidade

Este trabalho foi testado com dois outros grupos, todos utilizando o mesmo protocolo definido neste documento:

- Lucas Piske e Jeferson Batista
- Eric Marchetti e Vinicius Arruda

5 Conclusão

Neste trabalho foi visto a importância de um protocolo para a comunicação via rede. Também foi visto a dificuldade de se criar tal protocolo pensando em todos os problemas ao qual podem ocorrer.

Para a resolução deste problema foi utilizado a API protocol buffers do Google, que provém todo o serviço do protocolo apenas a partir da definição de uma estrutura a sua preferência e da utilização dos recursos da API. Foi notado que a curva de aprendizado para a utilização da API causa determinado atraso de produção porém pode ser recompensado após a integração da API de forma correta no projeto, gerando ganho de produção com a não preocupação de problemas que devem ser tratados na comunicação do protocolo, como por exemplo tamanho do arquivo a ser enviado pelo fluxo.

Em nosso trabalho foi implementado um servidor multi-thread ao qual permite a comunicação de vários clientes pela internet, onde pode ser testado a eficiência do protocolo buffer em casos reais, ao qual se tem aprovação para ser utilizado em nosso dia a dia.