

# Sistema de Escambo: KERO - SISTEMAS DISTRIBUÍDOS (CCF355)

(pt-4)

Cláudio Barbosa - 3492

Aryel Penido - 3500

Julho, 2022

## Contents

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Desenvolvimento</b>	<b>2</b>
2.1	Uso do gRPC . . . . .	2
2.2	Uso de Threads . . . . .	2
2.3	Servidor . . . . .	2
2.4	Cliente . . . . .	3
2.5	Protocol Buffers . . . . .	3
2.6	Interface gráfica . . . . .	3
2.7	Inicialização . . . . .	3
<b>3</b>	<b>Utilização do SD</b>	<b>4</b>
<b>4</b>	<b>Decisões de implementação</b>	<b>8</b>
<b>5</b>	<b>Conclusão</b>	<b>8</b>
5.1	Vantagens observadas . . . . .	8
5.2	Dificuldades e encontradas . . . . .	9
5.3	Considerações finais . . . . .	9
<b>6</b>	<b>Referências</b>	<b>9</b>

# 1 Introdução

Aprofundando e abstraindo o processo de desenvolvimento do nosso sistema distribuído, esta parte do trabalho consiste em implementar o sistema de escambo com o uso do middleware RMI gRPC. Ele atuara na comunicação entre os outrora processos, sendo implementados como objetos. A proposta é que com a utilização do gRPC o cliente consiga interagir com o servidor por meio de chamadas de funções simples, ou seja, de interfaces de códigos geradas automaticamente pela própria aplicação do gRPC. Já apontando uma das diferenças entre o uso de sockets e do gRPC é que as threads foram abstraídas com a utilização de um thread pool, bem como uma pool de processos. Retomo que os objetos de trocas escolhido pelo grupo são cervejas. A linguagem de programação escolhida pelo grupo foi **Python 3** e o banco de dados **SQLite** (pré-definido) implementando uma arquitetura cliente-servidor.

## 2 Desenvolvimento

Nesta seção estarão listados as decisões de implementação (estruturação), bem como detalhes importantes para o funcionamento do Sistema Distribuído (SD) idealizado pelo grupo.

### 2.1 Uso do gRPC

Como já dito neste trabalho, a ideia aqui é utilizar o gRPC propiciando uma interação cliente-servidor por meio de chamadas de funções simples, ou seja, de interfaces de códigos geradas automaticamente pela própria aplicação do gRPC. Isso significa que será necessário apenas implementar a lógica de programação, o que facilita muito a adoção desse recurso.

Um canal gRPC deve ser reutilizado ao fazer chamadas gRPC. Reutilizando um canal permite que as chamadas sejam multiplexadas por meio de uma conexão HTTP/2 existente. A abstração feita pelo gRPC segue basicamente a ordem abaixo:

- Abrindo um soquete
- Estabelecer conexão TCP
- Negociando TLS (O TLS (Transport Layer Security) é usado para proteger mensagens)
- Iniciando a conexão HTTP/2
- Fazendo a chamada gRPC

### 2.2 Uso de Threads

O uso da biblioteca **thread** foi substituído pela importação do pacote **futures** da biblioteca. Ela é necessária para a utilização de thread pool, que pode implementar uma sincronização tanto síncrona, como assíncrona. A maioria dos usuários desejará usar o modelo de sincronização: o servidor terá um pool de threads (interno) que gerencia solicitações de multiplexação em um certo número de threads (reutilizando threads entre solicitações). O modelo assíncrono permite que você traga seu próprio modelo de encadeamento, mas é um pouco mais complicado de usar - nesse modo, você solicita novas chamadas quando o servidor está pronto para elas e bloqueia as filas de conclusão enquanto não há trabalho a fazer. Ao organizar quando você bloqueia nas filas de conclusão e em quais filas de conclusão você faz solicitações, você pode organizar uma ampla variedade de modelos de encadeamento.

Ao contrário da parte anterior, não é necessário inicializar cada thread que será utilizada, esta parte é abstraída por meio do middleware.

### 2.3 Servidor

O uso de gRPC simplificou bastante o funcionamento do sistema. O servidor é inicializado seguindo a função abaixo. O número máximo de threads disponíveis é limitado a 10, isso implica em uma utilização de múltiplos clientes simultaneamente. A porta utilizada e conhecida foi a **"localhost:50051"**, que sera a mesma utilizada no lado do cliente. Assim o servidor estará rodando continuamente até que alguma forma de interrupção seja aplicada.

```
def servidor():  
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))  
    comunicacao_pb2_grpc.add_ComunicarServicer_to_server(ServidorKero(), server)
```

```
server.add_insecure_port("localhost:50051")
server.start()
server.wait_for_termination()
```

Um registro de requisições e replies é exibido enquanto o servidor estiver ativo. Isso possibilita um melhor acompanhamento de quais ações e quais usuários estão executando no sistema. Nosso servidor é responsável pelo acesso ao banco, sendo que o projeto foi idealizado de modo que todas as regras de negócios aplicáveis estejam encapsuladas no arquivo *rns.py*. O arquivo *servidor.py* é o local em que foram utilizadas esta lógica. O servidor foi reformulado e as rns sofreram alterações para que o sistema se comporte de maneira mais estável e apresente um desacoplamento da interface com o usuário, não mais sendo realizada comunicações entre estas classes.

## 2.4 Cliente

Agora o cliente é uma classe que contém os métodos de interface e comunicação que utilizam ops serviços fornecidos pelo servidor. Vale ressaltar que consultas ao banco e modificações seguem o fluxo de comunicação. Sendo que, em nosso sistema, é utilizada a comunicação unária por meio de requests(requisições do cliente) e replies (respostas do servidor). O cliente também é responsável pela interação mais amigável (interface) com o servidor.

```
stub = comunicacao_pb2_grpc.ComunicarStub(channel)
```

Do lado do cliente, temos uma chamada de rede feita por um stub, que é como um "falso" objeto representando o objeto do lado do servidor. Este objeto tem todos os métodos com suas assinaturas. o funcionamento é basicamente o mesmo, temos um cliente que converte as chamadas feitas localmente em chamadas de rede binárias com o protobuf e as envia pela rede até o servidor gRPC que as decodifica e responde para o cliente.

## 2.5 Protocol Buffers

Os protocol buffers são um método de serialização e desserialização de dados que funciona através de uma linguagem de definição de interfaces (IDL). A grande vantagem do protobuf é que ele não depende de plataforma, então o que fizemos foi escrever a especificação em uma linguagem neutra (o próprio proto) e compilar esse contrato para python, mas isso poderia ter sido feito para qualquer outra linguagem. O protobuf em si não contém nenhuma funcionalidade, ele é apenas um descritivo de um serviço, listando as funções e como as mensagens serão trocadas (estrutura). Com o que é esperado de parâmetro e o que cada função irá retornar.

O serviço no gRPC é um conjunto de métodos, pense nele como se fosse uma classe. Então podemos descrever cada serviços com seus parâmetros, entradas e saídas. Cada método (ou RPC) de um serviço só pode receber um único parâmetro de entrada e um de saída, por isso é importante podermos compor as mensagens de forma que elas formem um único componente, isso inclusive é um dos pontos limitantes observados no uso do gRPC. É sempre esperado um parâmetro de entrada, mesmo que ele seja vazio deve ser representado, como no caso abaixo:

```
rpc ListagemDeitensTroca (Vazia) returns (ListaCervejaBar);
```

Além disso, toda mensagem serializada com o protobuf é enviada em formato binário, de forma que a sua velocidade de transmissão para seu receptor é muito mais alta do que o texto puro, já que o binário ocupa menos banda e, como o dado é comprimido pelo HTTP/2, o uso de CPU também é muito menor.

## 2.6 Interface gráfica

Toda a interação entre usuário e sistema dará-se-a por meio de terminal. Os comandos e exemplos de como utilizar o SD serão abordados na seção de Utilização do SD. Basicamente o mesmo fluxo de ações do trabalho anterior foi aproveitado, com a realização de melhorias visuais. Algumas delas devem se a forma de representação de dados do próprio gRPC que permite, através do protobuffer, uma troca mais amigável entre cliente-servidor.

## 2.7 Inicialização

Como utilizaremos o SQLite, é necessário que ele esteja instalado para que o banco seja efetivamente criado e populado com os dados pré-definidos pelo grupo. Para tal basta utilizar o comando de instalação:

```
sudo apt install sqlite3
```

Alguns pontos devem ser observados para o uso do gRPC. Ele possui como pré-requisitos: Python 3.5 e pip versão 9.0.1 ou versões superiores. Talvez seja necessário atualizar a versão do pip, para tal poderá ser executado o comando:

```
python -m pip install --upgrade pip
```

Para a instalação global do gRPC:

```
sudo python -m pip install grpcio}
```

As ferramentas gRPC do Python incluem o compilador do protocolo de buffer e o plug-in especial para gerar código de servidor e cliente a partir de definições de serviço .proto. Para instalar as ferramentas gRPC, execute:

```
python -m pip install grpcio-tools
```

Para atualizar o código gRPC usado pelo sistema e implementado em comunicação.proto para que ele use as definições de serviços, espera-se que dois arquivos sejam criados (caso ausentes: comunicacao.pb2 e comunicacao.pb2 grpc) Na pasta raiz do trabalho, execute o seguinte comando:

```
python3 -m grpc_tools.protoc -I protos --python_out=.  
--grpc_python_out=. protos/comunicacao.proto
```

Para que o Servidor inicialize, é necessário utilizar o comando abaixo no terminal. O grupo **não** implementou interface gráfica interativa, sendo que toda a interação ocorrerá via terminal. No diretório do arquivo servidor.py, deverá ser executado o seguinte comando:

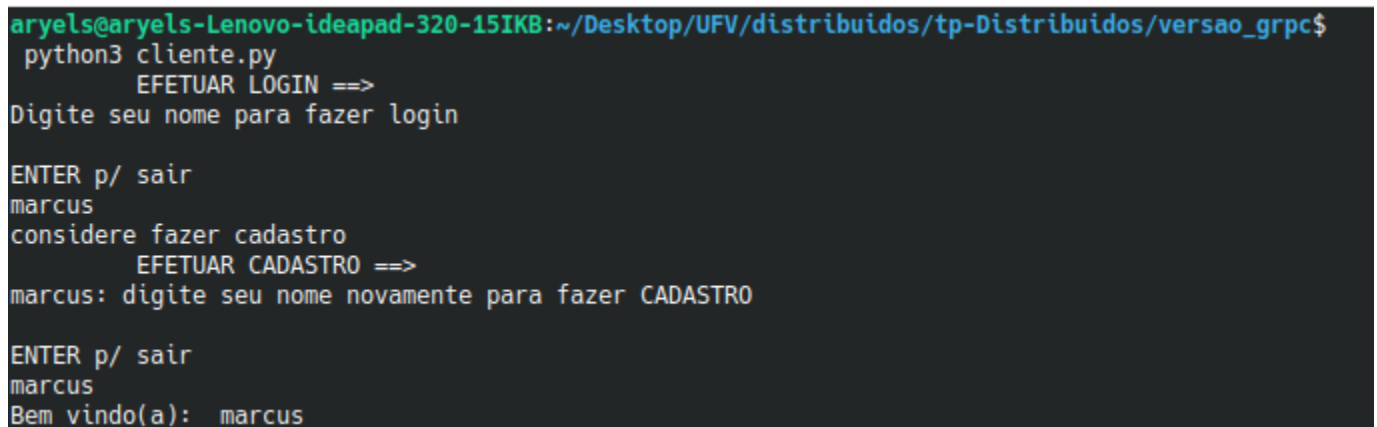
```
python3 servidor.py
```

O mesmo procedimento acima deverá ser realizado para que o(s) cliente(s) seja executado. Atentando-se para o fato que o diretório raiz do arquivo *cliente.py* é a pasta entidades:

```
python3 cliente.py
```

### 3 Utilização do SD

Assim como a versão anterior,, ao inicializar o servidor e instanciar um cliente é solicitado ao usuário que digite seu nome para logar no sistema. Caso nome não esteja no nosso banco de dados, é dada a opção de novo cadastro ao usuário.



```
aryels@aryels-Lenovo-ideapad-320-15IKB:~/Desktop/UFV/distribuidos/tp-Distribuidos/versao_grpc$  
python3 cliente.py  
EFETUAR LOGIN ==>  
Digite seu nome para fazer login  
  
ENTER p/ sair  
marcus  
considere fazer cadastro  
EFETUAR CADASTRO ==>  
marcus: digite seu nome novamente para fazer CADASTRO  
  
ENTER p/ sair  
marcus  
Bem vindo(a): marcus
```

Figure 1: Tela inicial

Quando logado no sistema, o menu principal é apresentado ao usuário, estão presentes todas as opções do sistema.

```

Bem vindo(a): marcus
MENU
1:BAR - itens p/ troca
2:Cadastrar cerveja
3:Ver sua Geladeira(Itens)
4:Enviar pedido de troca
5:Listar TROCAS PENDENTES
6:Aceitar/Recusar trocas
7:Kero Sair
ESCOLHA UMA DAS OPCOES:

```

Figure 2: Novo Menu Principal

É possível cadastrar uma nova cerveja, durante a execução dessa opção, ao usuário é solicitado que insira o nome, o abv, o ibu e o estilo da cerveja que será adicionada. São mostradas ainda **tooltips** com os significados das siglas ABV e IBU, buscando melhorar a experiência do usuário. Ele deverá escolher a opção "Cadastrar Cerveja"

```

CADASTRAR CERVEJA ==>
(SERVIDOR) < marcus > Vamos cadastrar uma cerveja

digite o nome cerveja
hoegaarden

Agora digite o ABV (ABV é a sigla para ALCOHOL BY VOLUME, ou quão alcoolico é um exemplar
4.9

Agora digite o IBU (IBU é a sigla para International Bitterness Units, ou o quão amargo é um e
xemplar
13

```

Figure 3: Cadastro de uma cerveja

```

Ta quaaaase! Digite agora o estilo
witbeer

Confirma as informações pra gente?

nome: hoegaarden ABV: 4.9 IBU: 13 estilo:witbeer

[S]im [N]ao
s
CLIENTE: message: "STATUS: 200"

Realizar outro cadastro??
[S]im [N]ao

```

Figure 4: Confirmação do cadastro de uma cerveja

Após a confirmação das informações do cadastro é dada a opção de repetir essa mesma operação e cadastrar mais uma cerveja

As trocas são feitas na opção **"4: Enviar pedido de troca"**, nela são listadas primeiro todas as cervejas disponíveis no bar para troca. Em seguida o usuário deve indicar o índice da cerveja que deseja solicitar. Após isso o usuário precisa indicar o índice da cerveja que deseja utilizar nesse troca.

```
}
cerveja {
  id: 12
  dono: "aryel"
  cerveja: "1"
  abv: "2"
  ibu: "3"
  estilo: "5"
}
cerveja {
  id: 13
  dono: "marcus"
  cerveja: "hoegaarden"
  abv: "4.9"
  ibu: "13"
  estilo: "witbeer"
}

aryel: Escolha primeiro a cerveja que você deseja solicitar!
É só digitar o índice da desejada
10
Veja sua geladeira novamente:
      SUA GELADEIRA ==>
cerveja {
  id: 3
  dono: "aryel"
  cerveja: "brahma"
  abv: "4.8"
  ibu: "18"
  estilo: "international lager"
}
cerveja {
  id: 12
  dono: "aryel"
  cerveja: "1"
  abv: "2"
  ibu: "3"
  estilo: "5"
}

^ Todos os itens listados | Vazio (Cadastre alguma cerveja)
aryel: Agora só escolher o índice da cerveja que você deseja dar em troca
3
Pedido de troca enviado
```

Figure 5: troca de uma cerveja

O menu **"5: Listar TROCAS PENDENTES"** mostra as solicitações de troca pendentes as quais o usuário pode responder.

```

MENU
1:BAR - itens p/ troca
2:Cadastrar cerveja
3:Ver sua Geladeira(Itens)
4:Enviar pedido de troca
5:Listar TROCAS PENDENTES
6:Aceitar/Recusar trocas
7:Kero Sair
  ESCOLHA UMA DAS OPCOES:5
    SUAS TROCAS ==>
troca {
  id: 2
  idCervejaOferecida: 1
  idCervejaDesejada: 3
  solicitante: "claudio"
  executor: "aryel"
}
^ Todas as trocas listadas | Vazio (Cadastre uma troca)
  MENU
  1:BAR - itens p/ troca
  2:Cadastrar cerveja
  3:Ver sua Geladeira(Itens)
  4:Enviar pedido de troca
  5:Listar TROCAS PENDENTES
  6:Aceitar/Recusar trocas
  7:Kero Sair
  ESCOLHA UMA DAS OPCOES:6
    RESPONDER TROCA ==>
    SUAS TROCAS ==>
troca {
  id: 2
  idCervejaOferecida: 1
  idCervejaDesejada: 3
  solicitante: "claudio"
  executor: "aryel"
}
^ Todas as trocas listadas | Vazio (Cadastre uma troca)
Digite o indice da troca que deseja responder
2
SHOW!Agora digite se deseja realizar a troca ou rejeitar a solicitação
[ (K)ero p/ aceitar | Qualquer coisa para rejeitar ]

```

Figure 6: Detalhes do processo de trocas de uma cerveja

O menu **"6: Aceitar/Recusar trocas"** permite que o usuário responda a solicitação de troca. Ele precisa inicialmente digitar o índice da troca que deseja atualizar e depois informar ao sistema se deseja aceitar a solicitação de troca ou a rejeitar.

Vale ressaltar o comportamento do servidor durante a execução do SD. Sempre que uma requisição é feita pelo cliente, uma mensagem é exibida no terminal do servidor informando qual request está sendo feita. Além disso, as respostas para essas requests também são mostradas nesse terminal.

```

aryels@aryels-Lenovo-ideapad-320-15IKB:~/Desktop/UFV/distribuidos/tp-Distribuidos/versao_grpc$
python3 servidor.py
SERVIDOR::Ativo!

Request Login
Resposta Login:  T
Request Login
Resposta Login:  F
Request Cadastro
Resposta Cadastro:
Request Cadastro Cerveja
veio banco
Resposta Cadastro:  200
Request Lista de Itens GELADEIRA
Request Lista de Itens BAR
Request Login
Resposta Login:  F
Request Cadastro
Resposta Cadastro:
Request Lista de Itens GELADEIRA
Request Lista de Itens BAR
Request Lista de Itens GELADEIRA
Request Troca de Cerveja
Resposta Troca:  200
Request Login
Resposta Login:  T
Request Lista de Itens GELADEIRA
Request Lista de Itens BAR
Request Lista de Itens GELADEIRA
Request Troca de Cerveja
Resposta Troca:  200
Request Login
Resposta Login:  T
Request Lista de Itens GELADEIRA
Request Lista de Itens BAR
Request Lista de Itens GELADEIRA
Request Troca de Cerveja
Resposta Troca:  200
Request Lista de TROCAS
Request Lista de TROCAS

```

Figure 7: Comportamento e movimentações no servidor

## 4 Decisões de implementação

Não foram realizadas alterações significativas que extrapolem a refatoração para utilização do middleware RMI (gRPC), sendo assim continuam válidas as decisões tomadas na parte 3.

## 5 Conclusão

### 5.1 Vantagens observadas

- Implementação relativamente fácil depois do desenvolvimento inicial;



- Melhor organização com a utilização do arquivo .proto, o que gera também uma padronização do código e garante o comportamento das chamadas de métodos;
- Um fluxo mais fluído entre cliente-servidor, sendo que muitos dos problemas de quebra do servidor quando alguma requisição de cliente ficava pendente e a conexão se encerrava foram sanadas;
- O uso de sockets, por possuir um nível baixo de abstração, produziu muito código e demandou um cuidado maior entre os processos, principalmente com o uso de threads;

## 5.2 Dificuldades e encontradas

- Uma curva de aprendizado maior, devido a complexidade maior que o funcionamento do socket;
- Foi possível perceber que a arquitetura de um sistema usando gRPC ficou um pouco mais complexa;
- A utilização do .proto com restrição de tipos de retorno múltiplos restringe um pouco as respostas das funções, o que não gera muita dificuldade pois esse problema pode ser sanado com o uso de tipos aninhados;

## 5.3 Considerações finais

Esse trabalho apresentou para o grupo um universo gigante de possibilidades de comunicações entre processos. Vale ressaltar o uso de protobuf que poderá ser estendido para outras áreas e é de grande utilização na parte de micro serviços. É interessante observar a melhoria que uma camada de abstração pode conceder a um sistema distribuído, mesmo em um simples como este de escambo. Muitas das possibilidades do uso do gRPC não foram abordadas e/ou aplicadas neste trabalho, bem como: streaming de mensagens (tanto de cliente, quanto de servidor e de ambos); chamadas totalmente assíncronas, e outras funções geradas automaticamente que são úteis em diversas áreas.

## 6 Referências

**Visão geral do gRPC.** <https://docs.microsoft.com/pt-br/aspnet/core/grpc/?view=aspnetcore-6.0>. Acesso em: 21 de julho de 2022.

**Quick start.** <https://grpc.io/docs/languages/python/quickstart/>. Acesso em: 21 de julho de 2022.

**O guia completo do gRPC parte 1: O que é gRPC?.** <https://blog.lsanatos.dev/guia-grpc-1/> Acesso em: 21 de julho de 2022.

**Python gRPC Tutorial - Create a gRPC Client and Server in Python with Various Types of gRPC Calls.** <https://www.youtube.com/watch?v=WB37L7PjI5kt=871s>. Acesso em: 21 de julho de 2022.

**Protocol Buffers Crash Course.** <https://www.youtube.com/watch?v=46O73On0gyI>. Acesso em: 21 de julho de 2022.