Redes de Computadores
Trabalho Prático - 1

João Pedro Schio Ortega

09 de Outubro de 2025

1 Descrição

O objetivo do trabalho é implementar e experimentar a comunicação em redes a partir da construção e da análise de servidores.

2 Servidor de chat multi-usuário

Para exercitar o uso de sockets, decidi implementar um servidor de chat onde clientes vão poder trocar mensagens globais ou criar grupos de conversa privada.

3 Implementação do servidor

A linguagem escolhida para o servidor foi Rust, devido às suas garantias de segurança e ao alto desempenho.

3.1 Compilar Rust

Como Rust ainda é uma linguagem mais nichada vou deixar aqui um tutorial de como compilar o servidor.
A maneira mais fácil e que funcionará independente da plataforma é utilizando o script rustup que a fundação Rust disponibilizou rustup.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Esse script vai identificar o seu sistema operacional e instalar a versão correspondente do sistema de compilação do rust.
Após isso, é preciso navegar para o diretório chat-server em um terminal e rodar os comandos:

```
cargo run # para rodar em debug
cargo run --release # para rodar em release
```

Ao rodar esses comandos, o cargo, (sistema de compilação do rust) começará a baixar as dependências do projeto. Para seguir as especificações do trabalho foram-se utilizadas apenas as crates (bibliotecas) tokio e serde-json
tokio serve para ter uma runtime assíncrona e serde-json serve para ter suporte a JSON.

3.2 Protocolo

O protocolo escolhido para o envio de mensagens foi TCP por conta de suas garantias de entrega

3.3 Api

Para manter o projeto o mais próximo ao hardware possível, não tentei implementar algo parecido com HTTP, utilizei apenas o envio de JSON via TCP.

3.4 As mensagens enviadas ao servidor

Todas as mensagens trocadas com o servidor são enviadas usando um arqvuivo JSON que contém os seguintes campos:

1. Tipo de mensagem
2. Usuário
3. Conteúdo

3.5 Tipos de mensagem

Os tipos de mensagem e seus comportamentos.

1. Ola
2. MensagemGlobal
3. MensagemPrivada
4. Tchau
5. GetMensagens

3.6 Comportamento das mensagens

Tipo	**Argumentos**	**Comportamento**
`Ola`	`String (nome)`	Adiciona o usuário à lista de conectados e envia uma mensagem de boas-vindas.
`Tchau`	`String (nome)`	Remove o usuário da lista de conectados e notifica os demais.
`MensagemPrivada`	`[String] (usuários), String (conteúdo)`	Envia uma mensagem privada apenas aos usuários especificados.
`MensagemGlobal`	`String (conteúdo)`	Envia uma mensagem a todos os usuários conectados.
`GetMensagens`	`String (Usuário)`	Recebe como resposta todas as mensagens globais e todas as mensagens privadas que o usuário faz parte

4 Cliente

Para fazer o cliente eu escolhi a linguagem de programação Python, por conta de sua simplicidade e facilidade de desenvolvimento.
Como a comunicação com o servidor é feita através de mensagens JSON enviadas via TCP o desenvolvimento do cliente foi simples, foi necessário apenas criar uma classe de comunicação com o servidor, uma função que desenha na tela as respostas do servidor e um switch case para cada tipo de mensagem que pode ser mandada.

4.1 Como usar o cliente

Nesta seção vou explicar o uso do cliente.
para roda-lo não é preciso instalar nenhuma biblioteca adicional, apenas o interpretador python.
Para rodar utilize o comando: python3 main.py .

4.1.1 Cadastro de usuário

O script perguntara seu nome de usuário, você pode digitar qualquer nome desde que não haja espaços.

4.1.2 Envio de mensagens globais

Para enviar mensagens globais, é necessário usar o comando:

```
/m (conteudo)
```

4.1.3 Envio de mensagens privadas

Para enviar mensagens privadas é necessário usar o comando

```
/mespriv (lista de nomes separados por virgula) (mensagem)
```

Chats privados podem conter dois ou mais integrantes.

5 Testes

Para testar o projeto, fiz um script em python que criará 1000 usuários, mandará 4000 mensagens globais, criará 500 chats privados com no minimo 2 pessoas e no máximo 50, e mandará no mínimo 100 mensagens em cada um desses chats privados. Resultado dos testes:

```
Resumo:
Usuarios:            1000 (reg ok=True, err=0)
Msgs globais:        meta 4000 (ok=4000, err=0)
Chats privados:      500
mensagens privadas   Meta 50000 (ok=50000, err=0)
python3 super_send.py  3.07s user 3.88s system 60% cpu 11.466 total
```

Como pode-se ver, o servidor conseguiu registrar toda essa carga em menos de 12 segundos, levando em consideração que existem alguns sleeps durante o código de teste, considero que o servidor de único thread assíncrono em Rust se deu muito bem.