Universidade Federal de Mato Grosso do Sul
Faculdade de Computação (FACOM)

Redes de Computadores

Prof. Hana Karina Salles Rubinsztejn

1° Trabalho Prático

Alberto Yoshiriki Hisano Higuti e Eduardo Lopes de Lemos

Outubro, 2024

1 Introdução

Este relatório descreve o desenvolvimento de um sistema de mensagens, inspirado no Twitter, proposto como parte da disciplina de Redes de Computadores na Universidade Federal de Mato Grosso do Sul. O objetivo principal do projeto é implementar um sistema funcional que permita o envio e recebimento de mensagens curtas em um ambiente multi cliente-servidor, utilizando comunicação via protocolo UDP. A implementação visa, além da funcionalidade, o domínio de técnicas de programação com sockets, processos, threads e temporização de mensagens.

O projeto consiste no desenvolvimento de dois componentes principais: o servidor, responsável pelo gerenciamento das mensagens e a comunicação entre os clientes, e o cliente, que será utilizado tanto para envio quanto para a exibição das mensagens. Seguindo as diretrizes do trabalho, o sistema é baseado em um protocolo de aplicação simplificado, com mensagens de estrutura definida para garantir a interoperabilidade entre as soluções dos diferentes grupos.

Este documento apresenta a arquitetura adotada, as escolhas de implementação e as funcionalidades desenvolvidas ao longo do projeto. Esse projeto pode ser acessado também no GitHub.

2 Visão Geral

Este projeto foi implementado em Python, utilizando a biblioteca *Streamlit* para criar uma interface gráfica interativa. A escolha de Python se deu por sua simplicidade e vasto suporte a bibliotecas para redes e interfaces, facilitando a implementação do servidor e cliente para troca de mensagens.

A interface gráfica construída com *Streamlit* permite que os usuários enviem e recebam mensagens de forma intuitiva, além de monitorar o estado do servidor em tempo real. Isso proporciona uma experiência de uso amigável e acessível.

O sistema utiliza o protocolo UDP para a comunicação entre os clientes e o servidor. As principais funcionalidades incluem o envio e recebimento de mensagens, além de atualizações periódicas do servidor sobre o número de clientes conectados e o tempo de execução.

Em resumo, o projeto combina a simplicidade do Python com a interatividade do *Streamlit* para fornecer um sistema de mensagens funcional e eficiente.

3 Arquitetura de Software e Servidor

A arquitetura geral do sistema segue os princípios da *"Clean Architecture"*, adotando todos os princípios *SOLID* para garantir uma clara separação de responsabilidades entre os componentes e a interface gráfica.

Essa abordagem permite que o *backend* seja completamente independente da interface de usuário, facilitando a modificação ou substituição da plataforma de visualização sem impactar a lógica de negócios.

No lado do servidor, a arquitetura é baseada no protocolo UDP, utilizando *threading* para a execução de tarefas periódicas e a função `select` para multiplexação de entrada/saída (E/S). Essa combinação permite um gerenciamento eficiente de múltiplos clientes, possibilitando a comunicação assíncrona para o envio e recebimento de mensagens.

O servidor utiliza *sockets* UDP para a comunicação com os clientes, com as funções `sendto` e `recvfrom` para o envio e recebimento de mensagens. A função `select` monitora simultaneamente múltiplos clientes, identificando quando há dados disponíveis para leitura sem bloquear o programa.

Para o processamento de mensagens, o servidor utiliza a função `handle_message`, que despacha as mensagens com base no tipo. As mensagens podem ser de saudação (OI), saída ($TCHAU$), texto (MSG) ou erro ($ERRO$). Cada tipo é tratado por um manipulador específico, assegurando que as ações corretas sejam tomadas de acordo com o conteúdo da mensagem recebida.

Além disso, o servidor implementa a função `periodic_status_message`, que envia atualizações de status a cada 60 segundos, informando o número de clientes conectados e o tempo de atividade do servidor. A função `run` inicia uma *thread daemon* para garantir que o servidor permaneça responsivo, enquanto processa mensagens e realiza tarefas em segundo plano.

4 Refinamentos e Estrutura de Dados

O servidor e o cliente mantêm uma estrutura de dados baseada em um dicionário Python, onde as chaves representam os *client_ids* e os valores contêm as mensagens associadas. As mensagens são representadas pela classe `TwotterMessage`, que define todos os campos necessários para a comunicação.

O processo de codificação e decodificação das mensagens é feito por funções comuns a todos os componentes do sistema, `encode_message` e `decode_message`, permitindo que tanto o cliente quanto o servidor compartilhem o mesmo formato de mensagens. Isso facilita a manutenção e a adição de novos campos ou funcionalidades, bastando alterar uma única parte do código.

As mensagens são manipuladas como objetos no servidor e no cliente, garantindo acesso rápido a seus atributos, o que simplifica o tratamento das mensagens recebidas e enviadas. Essa abordagem centralizada de codificação e decodificação melhora a escalabilidade do sistema e simplifica sua manutenção.

As principais decisões de implementação incluem:

- A utilização de *threading* para garantir o envio de mensagens de status periódicas sem interromper a comunicação com os clientes.
- A separação completa entre o *backend* e a interface gráfica, permitindo que qualquer ferramenta ou plataforma de visualização possa ser utilizada sem impactar a lógica do servidor.
- A centralização do processo de codificação e decodificação das mensagens, garantindo facilidade de manutenção e escalabilidade ao sistema.

Essa arquitetura e refinamento na estrutura de dados garantem que o sistema funcione de maneira assíncrona, eficiente e escalável, facilitando a comunicação entre múltiplos clientes de forma rápida e precisa, enquanto o servidor realiza tarefas de manutenção em segundo plano.

5 Funcionalidades Extras

O sistema desenvolvido inclui algumas funcionalidades extras que melhoram a experiência do usuário. Uma dessas funcionalidades é a possibilidade de enviar mensagens para si mesmo, permitindo que o cliente utilize o sistema como um "bloco de notas". O envio dessas mensagens é tratado de forma eficiente, sem causar duplicação ou erros, garantindo que o usuário possa armazenar suas próprias anotações sem interferência no fluxo de comunicação com outros clientes.

Além disso, o nome de usuário no sistema é representado por um emoji, oferecendo uma maneira divertida e personalizada de identificação. O cliente pode escolher seu emoji a partir de uma lista de emojis disponíveis, permitindo uma forma rápida e visual de se diferenciar dos outros usuários conectados ao servidor.

Finalmente, a interface desenvolvida com o uso de Streamlit exibe em tempo real a lista de clientes exibidores ativos no sistema. O servidor mantém essa lista atualizada, e o cliente pode visualizá-la na tela constantemente durante a conversa, sem precisar solicitar explicitamente. Isso facilita a interação contínua entre os participantes, permitindo que o usuário veja facilmente quem está conectado em qualquer momento da sessão.

6 Setup

Para instalar as dependências do projeto, bastar rodar os comandos abaixo:

```
pip install poetry
git clone https://github.com/edu010101/Twotter.git
cd Twotter
poetry install
```

Caso esteja sendo utilizado o Sistema Operacional Windows será necessário criar um ambiente virtual, e após isso instalar as dependências do projeto. Isso pode ser feito com os seguintes comandos:

```
python -m venv trab_redes
trab_redes\Scripts\activate
pip install poetry
git clone https://github.com/edu010101/Twotter.git
cd Twotter
poetry install
```

7 Execução

Para executar este trabalho será necessário utilizar dois terminais, um para o servidor e outro para o cliente, da seguinte forma:

1. **Servidor**
   No terminal do servidor o seguinte comando deverá ser executado:

   ```
   poetry run python start_server.py
   ```

2. **Cliente**
   Para o terminal do cliente deve-se rodar a instrução a seguir:

   ```
   poetry run streamlit run start_client.py
   ```

8 Conclusão

Portanto, o desenvolvimento deste sistema de mensagens utilizando Python e *Streamlit* atingiu os objetivos propostos, permitindo a troca eficiente de mensagens em um ambiente multi cliente-servidor via protocolo UDP. A implementação demonstrou a capacidade de lidar com múltiplos clientes de forma simultânea, com envio e recebimento de mensagens em tempo real. A interface gráfica criada com *Streamlit* proporcionou uma interação amigável e acessível para os usuários.