TRABALHO SERVIDOR-CLIENTE UDP

TÍTULO: TWITTER AVANÇADISSIMO

Integrantes:

- Carlos Eduardo da Silva Trindade
- Matheus Bernard Mota
- Thiago Aparecido

Ferramentas:

- Linguagem Python

Bibliotecas importadas:

- time
- socket
- threading
- tkinter
- typing
- struct
- datetime

Utilidades Extras:

- User Experience, com uma interface intuitiva, facilidade de alteração de configurações de rede para facilidade de correção e cores que denotam o status de conexão, tal como pop-ups que direcionam o usuário ao uso correto da aplicação e caixas que são desabilitadas e só podem ser preenchidas nas condições corretas

Como Executar

Dentro do projeto, na pasta trabalho-python, há as pastas client e server. é possível executar o código a partir dos seguintes comandos

```Python
python server/server_main.py
python client/client_main.py
```

Pode ser necessário baixar algumas bibliotecas utilizando **pip install**

Resumo

Foi utilizado a biblioteca sockets para criar dois pontos que se conectam à rede local.

O socket servidor se encontra no endereço IP ‘0.0.0.0’ para que possa receber dados de todos os canais de rede, e o socket cliente se encontra no endereço IP‘127.0.0.1’, o endereço de loopback, que pode ser facilmente alterado.

A porta para ambos é 12345

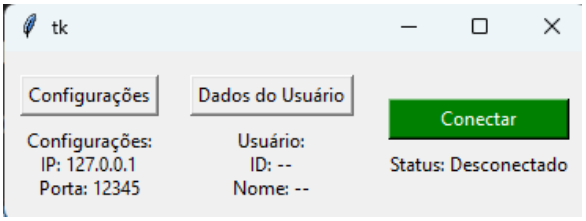
O projeto foi separado em dois subprojetos, um para o client e outro para o server.

Composição

Cliente

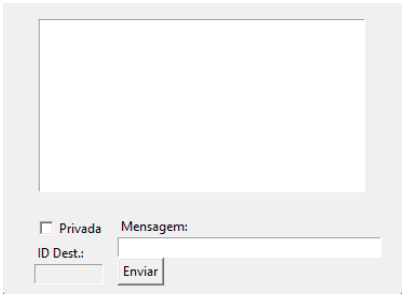
O projeto do cliente está dividido da seguinte forma:

- O módulo **client_utils** contém as constantes e métodos de utilidade e formalização de envio, ele contém os seguintes componentes:
  - métodos **sendOi**, **sendMsg** e **sendTchau**, cuja função é agrupar os dados da mensagem com as configurações corretas e facilitar seu uso no código principal
  - constantes frequentemente utilizadas, como o endereço do servidor, os valores dos tipos de mensagem, etc.
- O módulo **connection_pannel** contém a classe **ConnApp,** que é responsável pela interface gráfica do cliente e por controlar a conexão, mandando **sendOi** e **sendTchau** para o servidor e controlando os dados do usuário, tal como IP e Porta



- O módulo **messaging_pannel** contém a classe **MessagingApp,** que é responsável por exibir as mensagens. Esta classe recebe **ConnApp** como parâmetro para poder controlar o envio de mensagens tendo acesso aos dados do usuário e da conexão.
  - As mensagens são recebidas com uma thread que roda o método **message_receiver,** que recebe as mensagens em tempo real.

- O módulo **client_main** importa os módulos e é responsável pela execução do sistema



Servidor

O projeto do servidor está dividido da seguinte forma:

- Módulo server_utils:
    - Contém os métodos **sendOi, sendMsg e sendTchau,** com diferenças dos métodos de mesmo nome do cliente para se adequar às necessidades do servidor.
    - Contém as constantes utilizadas para definição das configurações de envio e de rede do sistema
    - Contém a função **make_standard_message,** que é utilizada para o envio periódico de mensagens
- Módulo server_main:
    - **active_clients:** lista de clientes ativos, para controle de clientes
    - **thread_sender_loop:** É responsável pelo envio periódico de mensagens, rodando em paralelo com uma thread separada
    - **handle_recv:** É o principal método, uma thread se separa do loop principal para responder todos os envios de mensagens recebidos pelos clientes, adicionando um cliente, removendo-o ou re-direcionando corretamente a mensagem recebida.
    - **startServer:** É a função que inicia o servidor, utilizando uma thread com timeout de 3 segundos, um loop roda **recvfrom** e abre uma thread para lidar com o recebimento.
    - **endServer:** Quando o servidor fecha, uma mensagem de tchau é enviada a todos os clientes e espera as threads acabarem
    - **return_error e validate_msg :** Utilizada para validar a mensagem recebida em **handle_recv**