Pontifícia Universidade Católica do Paraná (PUCPR)
NOMES: Lucas Moro, Juliana Xavier
Segundo Período

PROJETO CHAT
Curitiba, Paraná
2024

Introdução

Este projeto visa a criação de um sistema de chat que permite comunicação pública e privada entre clientes conectados a um servidor central.
A implementação foca na criação de um servidor e um cliente com interface gráfica amigável.
Para isso, foram utilizadas bibliotecas como socket para comunicação de rede, threading para gerenciamento de fluxos simultâneos e PyQt5 para a interface gráfica do cliente.

Estrutura do Servidor

O servidor foi implementado utilizando uma abordagem orientada a objetos, estruturado na classe ChatServer.
Essa classe organiza a lógica e as funções principais, facilitando o gerenciamento das conexões, envio de mensagens e manutenção de usuários conectados.
Abaixo, estão descritos os principais métodos e funcionalidades do servidor:

Inicialização do Servidor (__init__)

No construtor da classe ChatServer, são definidos o endereço IP e a porta para o servidor (host e port).

É configurado o socket para aceitar múltiplas conexões simultâneas e a reutilização de endereços com SO_REUSEADDR.

Também são criadas listas e dicionários para armazenar clientes conectados e associar cada cliente ao seu nome de usuário.

Função de Broadcast (broadcast)

Este método é responsável por enviar mensagens a todos os clientes conectados, exceto o remetente da mensagem.

Caso algum cliente não possa receber a mensagem, ele é removido da lista de clientes ativos para manter a integridade do sistema.

Tratamento de Clientes (handle_client)
Cada cliente é gerenciado em uma thread separada usando o método handle_client. Esse método distingue mensagens públicas e privadas.

Para mensagens privadas, o método busca o destinatário no dicionário de nomes de usuário e utiliza send_private_message para enviar a mensagem exclusivamente ao destinatário.

Em caso de erro, o cliente é removido e os demais usuários são notificados.

Envio de Mensagens Privadas (send_private_message)
Este método permite que um cliente envie uma mensagem privada para outro usuário. Ele verifica se o destinatário existe no dicionário de nomes e, em caso de sucesso, envia a mensagem tanto para o destinatário quanto uma confirmação para o remetente.

Se ocorrer uma falha, o cliente destinatário é removido.

Remoção de Clientes (remove_client)
A função remove_client é responsável por desconectar um cliente em caso de erro ou saída voluntária. Ela atualiza a lista de clientes e remove o nome de usuário do dicionário, além de enviar uma notificação aos demais usuários sobre a saída do cliente.

Aceitação de Conexões (accept_connections)
Este método espera novas conexões de clientes. Quando um cliente se conecta, o servidor solicita um nome de usuário e verifica sua disponibilidade.

Caso o nome já esteja em uso, a conexão é encerrada. Para clientes aceitos, o método armazena o cliente e seu nome de usuário e envia uma notificação a todos os usuários sobre a nova conexão.

Em seguida, cria uma thread para gerenciar as mensagens do cliente.

Inicialização do Servidor (start)
Este método inicia o servidor e chama accept_connections para começar a aceitar clientes. Esse fluxo permite que o servidor funcione continuamente,

aceitando novas conexões e gerenciando a comunicação entre os clientes.

Esse design orientado a objetos proporciona um código modular, fácil de manter e estender. A separação de responsabilidades em métodos distintos facilita o gerenciamento de conexões e a comunicação eficiente entre os clientes.

Estrutura do Cliente
O cliente foi implementado utilizando a biblioteca PyQt5 para fornecer uma interface gráfica intuitiva, permitindo que os usuários interajam facilmente com o chat.
A classe principal ChatClientGUI gerencia tanto a interface quanto a

comunicação com o servidor. A seguir, detalhamos as principais funcionalidades e métodos:

Inicialização do Cliente (__init__)
O construtor da classe ChatClientGUI inicializa o socket do cliente para comunicação TCP/IP e define atributos básicos, como o nome de usuário (self.username) e uma flag de execução (self.running).

Em seguida, chama o método init_ui() para configurar a interface gráfica.

Configuração da Interface Gráfica (init_ui)
Este método configura a janela principal do cliente, definindo o título, tamanho e layout.

São criados os seguintes componentes: QTextEdit (self.chat_display), QLineEdit (self.entry_message), e QPushButton (send_button).

Os componentes são adicionados a um layout vertical (QVBoxLayout) para organização na interface.

Conexão com o Servidor (connect_to_server)
Este método tenta estabelecer conexão com o servidor utilizando o IP e a porta especificados (server_ip e server_port).

Após a conexão bem-sucedida, chama prompt_username(), envia o nome de usuário ao servidor, exibe uma mensagem de confirmação na interface e inicia uma thread para receber mensagens de forma assíncrona.

Solicitação de Nome de Usuário (prompt_username)
Utiliza QInputDialog para solicitar ao usuário que insira um nome de usuário. Verifica se o nome não está vazio.

Caso esteja, exibe um aviso e solicita novamente, garantindo que um nome válido seja fornecido.

Recebimento de Mensagens (receive_messages)
Executado em uma thread separada, este método permanece em um loop enquanto self.running for True, recebendo e exibindo as mensagens do servidor.

Em caso de erro ou desconexão, fecha o socket do cliente e interrompe o loop.

Exibição de Mensagens (display_message)
Este método adiciona a mensagem recebida ao QTextEdit (self.chat_display) e garante que a última mensagem esteja visível utilizando ensureCursorVisible().

Isso mantém o histórico do chat atualizado e facilita a visualização pelo usuário.

Envio de Mensagens (send_message)
Quando o usuário envia uma mensagem (clicando no botão ou pressionando Enter), este método lê o texto do campo de entrada, verifica se é "/exit" e,
se sim, encerra a conexão e fecha a interface. Caso contrário, envia a mensagem ao servidor.

Evento de Fechamento da Janela (closeEvent)
Este método é chamado automaticamente quando o usuário tenta fechar a janela. Ele garante que a conexão seja encerrada corretamente,
enviando uma mensagem de saída ao servidor e fechando o socket.

Inicialização da Interface do Cliente (start_client_interface)
A função start_client_interface() inicializa a aplicação PyQt, cria uma instância de ChatClientGUI,
conecta-se ao servidor e exibe a interface gráfica.

Esse design orientado a objetos permite uma separação clara entre a lógica de negócios e a interface gráfica, facilitando futuras expansões e manutenção do código.

Conclusão
O sistema de chat desenvolvido oferece uma interface amigável para o usuário final e uma estrutura de servidor capaz de lidar com múltiplos clientes e mensagens simultâneas.
A aplicação das bibliotecas socket, threading e PyQt5 permitiu uma implementação eficiente da comunicação em rede e da interface gráfica.

A transição para uma abordagem orientada a objetos tanto no servidor quanto no cliente proporcionou um código mais organizado, modular e de fácil manutenção.
A separação de responsabilidades em classes e métodos específicos torna o sistema escalável e preparado para futuras melhorias.

Futuras implementações podem incluir:
- Autenticação de Usuários: Implementar um sistema de login com autenticação para garantir que apenas usuários registrados possam acessar o chat.
- Criptografia de Mensagens: Adicionar criptografia nas mensagens trocadas para aumentar a segurança e privacidade dos usuários.
- Suporte a Grupos e Salas de Chat: Permitir a criação de salas de chat ou grupos para segmentar conversas entre diferentes conjuntos de usuários.
- Notificações e Interface Aprimorada: Melhorar a interface gráfica com recursos

adicionais, como notificações de novas mensagens ou personalização do tema.

Justificativa da Abordagem Orientada a Objetos para a Interface Gráfica e o Projeto

Uso da Orientação a Objetos na Interface Gráfica

A interface gráfica do cliente foi implementada utilizando a biblioteca PyQt5, que é baseada em uma estrutura orientada a objetos (OO). Optamos por essa abordagem devido às vantagens que a OO oferece ao desenvolver interfaces gráficas, especialmente em termos de modularidade, manutenção e estrutura do código. As principais razões para essa escolha são:

1. **Modularidade e Reutilização de Código**
   A programação orientada a objetos permite a organização dos elementos visuais e lógicos (botões, caixas de texto, eventos) em classes, tornando o código mais modular. Por exemplo, a classe ChatClientGUI encapsula todos os componentes e a lógica de interação com o servidor, facilitando a localização e reutilização de cada elemento da interface.

2. **Facilidade de Extensão e Manutenção**
   A OO facilita a adição de novos recursos. Por exemplo, para expandir as funcionalidades do cliente, como adicionar uma área de notificações, basta estender a classe existente ou criar uma nova classe com base nela. Essa abordagem reduz o risco de conflitos e torna o código mais compreensível e fácil de manter.

3. **Desacoplamento da Lógica e Interface Gráfica**
   Com a OO, é mais simples separar a lógica de negócios (por exemplo, envio e recebimento de mensagens) da lógica de apresentação (exibição das mensagens). A ChatClientGUI concentra os métodos e atributos relacionados à interface gráfica, permitindo que o código de comunicação com o servidor permaneça separado e organizado.

4. **Uso de Callbacks e Manipuladores de Eventos**
   A interface gráfica é baseada em eventos (por exemplo, cliques em botões, entradas de texto). Com OO, é possível definir métodos específicos para lidar com esses eventos dentro da classe correspondente, tornando o código mais intuitivo e organizado. No caso do cliente, o método send_message é fácil de encontrar e modificar, pois está claramente associado à funcionalidade de envio de mensagens dentro da classe ChatClientGUI.

5. **Compatibilidade com a Estrutura da Biblioteca PyQt5**
   A biblioteca PyQt5 já é projetada com uma estrutura orientada a objetos, com componentes como QWidget, QPushButton, e QLineEdit, que permitem personalizações ao serem herdados. A utilização de classes permite aproveitar ao máximo as funcionalidades oferecidas pela biblioteca, integrando e organizando os elementos de forma eficiente.

Abordagem Orientada a Objetos no Projeto como um Todo

Além de ser útil para a interface gráfica, a orientação a objetos foi escolhida para o projeto completo por suas vantagens em termos de organização, escalabilidade e manutenção do código. O design orientado a objetos foi aplicado tanto ao servidor quanto ao cliente para os seguintes benefícios:

1. **Organização e Estruturação do Código**
   A abordagem orientada a objetos permite que as funcionalidades do sistema de chat sejam separadas em classes, cada uma com responsabilidades específicas. No servidor, a classe ChatServer centraliza a lógica de comunicação e gerenciamento de clientes, facilitando a compreensão do código e permitindo que os desenvolvedores saibam exatamente onde estão definidas as principais funcionalidades.

2. **Escalabilidade**
   A OO facilita a expansão do sistema. Caso novas funcionalidades sejam adicionadas ao servidor (como autenticação de usuários ou suporte a grupos de chat), basta estender a classe ChatServer ou adicionar novas classes que interajam com ela. Isso torna o sistema mais escalável e preparado para evoluir com novas demandas.

3. **Reutilização e Extensão de Funcionalidades**
   A utilização de classes permite que métodos e atributos sejam herdados, facilitando a extensão das funcionalidades existentes sem a necessidade de duplicar código. Caso o projeto seja ampliado para incluir novas interfaces ou funcionalidades de mensagens, será possível reutilizar e adaptar as classes existentes.

4. **Facilidade de Manutenção**
   Com a OO, o código é mais modular e dividido em responsabilidades específicas, facilitando a localização de problemas e a manutenção geral. No caso do sistema de chat, se algum problema ocorre no gerenciamento de conexões, a equipe de desenvolvimento sabe que precisa verificar a

classe ChatServer, enquanto problemas na interface gráfica podem ser resolvidos na classe ChatClientGUI.

5. **Adoção de Padrões de Projeto**
   A OO permite a utilização de padrões de projeto, como Singleton, Observer e Factory, que são úteis para organizar a lógica de aplicações complexas. A estrutura orientada a objetos deixa o projeto mais preparado para adotar esses padrões, caso sejam necessários.

Conclusão

Optar pela programação orientada a objetos no desenvolvimento do sistema de chat, tanto para o servidor quanto para o cliente, trouxe inúmeras vantagens, como uma melhor organização, facilidade de expansão e manutenção. A separação entre lógica de comunicação e de apresentação foi essencial para a clareza e modularidade do código, permitindo que o sistema seja mais robusto, escalável e fácil de manter.