Disciplina: Redes de Computadores - 2024/2
Aluna: Ana Tereza Ribeiro Soares Pereira
Professor: Magnos Martinello

Relatório de Implementação do Servidor de Chat Multissala

1. Escolhas de Implementação

a) Arquitetura Cliente-Servidor

O projeto utiliza uma arquitetura baseada em sockets TCP, que fornece uma comunicação confiável entre cliente e servidor. O servidor escuta conexões em uma porta fixa e aceita múltiplos clientes, encaminhando mensagens para a sala apropriada. Esse modelo garante a separação das funcionalidades e permite que cada cliente receba as mensagens corretamente sem interferência de outras salas.

b) Uso de Threads para Conexões Simultâneas

Cada cliente é gerenciado em uma thread separada. Isso permite que o servidor continue aceitando novas conexões enquanto trata as mensagens de usuários já conectados. Essa abordagem simplifica a implementação, mas pode gerar sobrecarga em cenários de alta concorrência.

c) Gerenciamento de Salas e Usuários

Foram utilizadas estruturas de dicionário para armazenar informações sobre as salas e os clientes conectados:

- `rooms`: dicionário que mapeia nomes de salas para uma estrutura contendo a lista de clientes, o criador da sala e, se necessário, uma senha de acesso.
- `clients`: mapeia os sockets para os respectivos usuários, armazenando nome e cor de identificação.
- `logs`: mantém um histórico de mensagens enviadas para facilitar a auditoria e depuração.

O uso de dicionários garante acessos rápidos (O(1) em média), tornando a busca e manipulação de dados eficiente.

d) Comandos de Controle e Interação

A aplicação implementa comandos que permitem aos usuários interagir de maneira mais eficiente com o servidor:

- `/sair`: remove o usuário da sala atual e retorna ao menu de seleção de salas.
- `/encerrar`: fecha a conexão do usuário com o servidor.
- `/quem`: lista os usuários conectados na sala atual.
- `@usuário mensagem`: permite enviar mensagens privadas.

Esses comandos garantem uma melhor experiência do usuário e tornam a aplicação mais interativa e funcional.

2. Complexidade e Otimização

a) Complexidade Temporal

- **Conexão e recepção de mensagens:** O(1), pois cada mensagem é recebida e enviada diretamente pelo socket.
- **Busca por usuários e salas:** O(n), pois pode ser necessário percorrer a lista de usuários em uma sala ou o dicionário de salas.
- **Envio de mensagens para todos os usuários da sala:** O(n), onde n é o número de usuários na sala.
- **Gerenciamento de clientes e encerramento:** O(1), pois as remoções do dicionário são eficientes.

b) Possíveis Otimizações

- **Uso de conjuntos (`set()`)** para armazenar usuários dentro das salas, otimizando buscas e evitando duplicações.
- **Filas assíncronas (`queue.Queue`)** para o envio de mensagens, reduzindo o risco de bloqueios entre threads.
- **Alternativa com `asyncio`**, permitindo que o servidor utilize I/O assíncrono em vez de múltiplas threads, melhorando a escalabilidade.

3. Melhorias Futuras

a) Melhorias na Estrutura do Código

- Separar o código do servidor em módulos distintos para maior organização (exemplo: `servidor.py`, `cliente.py`, `utils.py`).
- Melhorar o tratamento de exceções para evitar falhas inesperadas e melhorar a robustez do sistema.
- Adicionar logs mais detalhados, permitindo depuração mais eficiente e rastreamento de atividades.

b) Novas Funcionalidades

- **Histórico de mensagens por sala:** permitir que novos usuários visualizem as últimas mensagens ao entrar.
- **Notificações de entrada e saída em tempo real:** destacar quando um usuário entra ou sai da sala.
- **Recurso de moderação:** permitir que o criador da sala expulse usuários indesejados.
- **Sistema de reconexão automática:** permitir que usuários retornem à sala caso percam a conexão temporariamente.