

# Sistema de Chat Distribuído (Multi-Cliente)

**Equipe: Bernardo Heuer, Silvio Fittipaldi, Rodrigo Nunes, Eduardo Roma, Ronaldo Souto, Luís Melo**  
**Turma A, CESAR School, Recife - PE**

## RESUMO

Este artigo descreve o projeto e a implementação de um sistema de chat multi-cliente, desenvolvido como uma solução prática para aplicar conceitos de sistemas distribuídos, concorrência e sincronização. Utilizando a linguagem Python e suas bibliotecas nativas socket e threading, foi construída uma aplicação na arquitetura cliente-servidor capaz de gerenciar a comunicação simultânea de múltiplos usuários em tempo real. O servidor adota um modelo concorrente de uma thread por cliente para garantir responsividade, enquanto o acesso a recursos compartilhados, como a lista de clientes conectados, é protegido por mecanismos de Lock para prevenir condições de corrida e garantir a consistência dos dados. O resultado é um sistema funcional e robusto que demonstra com sucesso os princípios teóricos estudados.

## 1. INTRODUÇÃO

Sistemas distribuídos são uma peça fundamental da computação moderna, permitindo que aplicações alcancem alta escalabilidade, tolerância a falhas e compartilhamento de recursos ao distribuir a carga de processamento e dados entre múltiplos nós interconectados. Nesse contexto, os conceitos de concorrência e paralelismo são essenciais, pois permitem que um sistema execute múltiplas tarefas de forma simultânea, como atender a requisições de vários usuários ao mesmo tempo.

A complexidade destes sistemas reside não apenas na comunicação entre seus componentes, mas também no gerenciamento de recursos compartilhados. Quando múltiplas threads ou processos tentam acessar ou modificar o mesmo dado concorrentemente, podem surgir problemas como as condições de corrida (*race conditions*), que levam a estados inconsistentes e comportamento imprevisível. Portanto, a utilização de mecanismos de sincronização é crucial para garantir a integridade e a corretude da aplicação.

O objetivo deste projeto foi desenvolver uma solução prática que materializasse esses conceitos. Para isso, foi implementado um sistema de chat em tempo real, uma aplicação que, por sua natureza, exige uma arquitetura distribuída para conectar diferentes usuários e um forte modelo de concorrência para gerenciar suas interações simultâneas.

## 2. METODOLOGIA

A solução foi desenvolvida utilizando a linguagem Python 3, devido à sua simplicidade e ao seu robusto suporte a programação de rede e concorrência através de bibliotecas padrão.

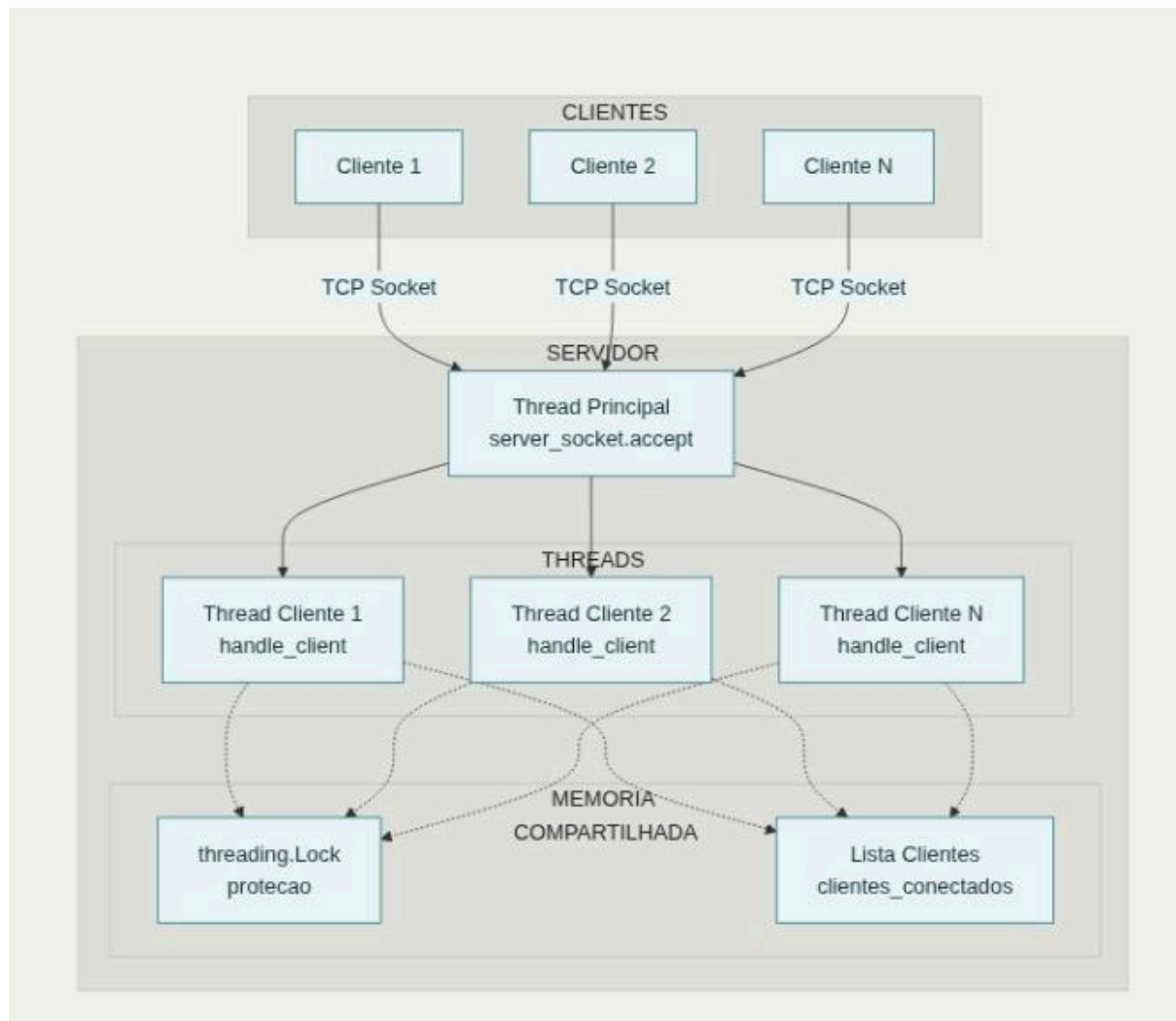
### 2.1. ARQUITETURA DA SOLUÇÃO

Foi adotada a arquitetura Cliente-Servidor, um modelo distribuído clássico e eficaz para aplicações centralizadas.

- **Servidor (server.py):** Atua como o nó central do sistema. Suas responsabilidades são: aguardar e aceitar novas conexões de clientes; gerenciar a lista de clientes ativos; receber as mensagens de um cliente e transmiti-las (broadcast) para todos os outros clientes conectados.
- **Cliente (client.py):** Representa o ponto de interação do usuário. Suas responsabilidades são: estabelecer uma conexão com o servidor; enviar as mensagens digitadas pelo usuário; e receber e exibir as mensagens vindas do servidor (sejam de outros clientes ou notificações do sistema).

### 2.2. DIAGRAMA DA ARQUITETURA

A Figura 1 ilustra a arquitetura geral do sistema, destacando a interação entre os componentes e o gerenciamento de concorrência no servidor.



**Figura 1 – Diagrama da Arquitetura Cliente-Servidor do Sistema de Chat**

### **2.3. IMPLEMENTAÇÃO DA CONCORRÊNCIA**

Para que o servidor pudesse atender a múltiplos clientes simultaneamente sem que um bloqueasse o outro, foi implementado o modelo thread-per-client. A thread principal do servidor fica em um loop infinito, responsável apenas por aceitar novas conexões com o método “server\_socket.accept()”. Para cada nova conexão aceita, uma nova thread (threading.Thread) é criada e iniciada, delegando a ela toda a comunicação com aquele cliente específico através da função “handle\_client”.

### **2.4. SINCRONIZAÇÃO DE RECURSOS COMPARTILHADOS**

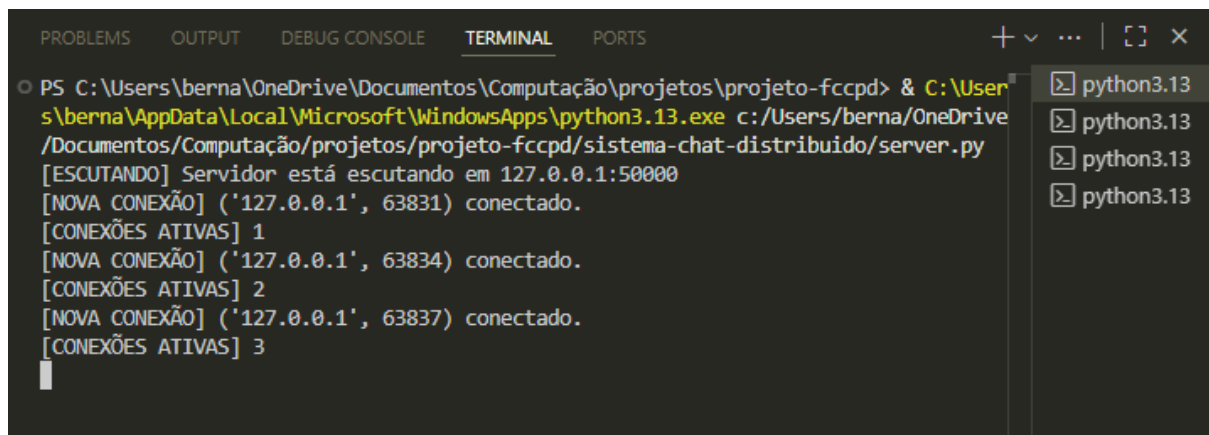
A lista de sockets dos clientes conectados (clientes\_conectados) é um recurso compartilhado entre todas as threads ativas no servidor. Uma operação de leitura por uma thread enquanto outra escreve poderia levar a erros e inconsistências. Para resolver este problema, foi utilizado um objeto “threading.Lock”. Toda e qualquer operação sobre esta lista (adicionar um novo cliente, remover um cliente desconectado ou iterar sobre ela para enviar uma mensagem de broadcast) é executada dentro de um bloco “with lock:”. Isso

garante que as operações sejam atômicas, ou seja, apenas uma thread pode acessar a lista por vez, prevenindo condições de corrida e garantindo a integridade dos dados.

### 3. RESULTADOS

O sistema foi testado em um ambiente local (localhost), onde o servidor foi executado em um terminal e múltiplos clientes foram executados em terminais separados para simular uma interação real.

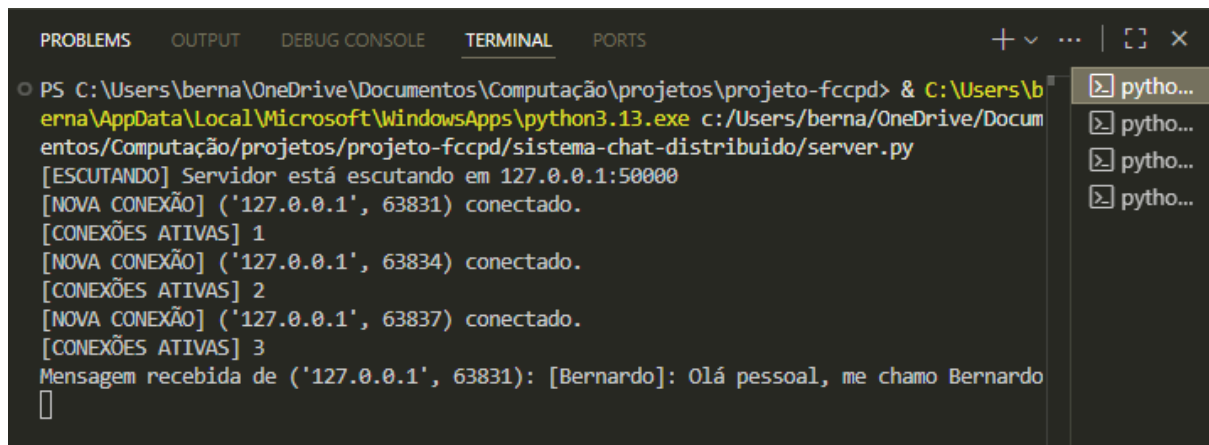
A Figura 2 mostra o terminal do servidor no momento em que ele está ativo e após três clientes se conectarem. É possível observar as mensagens de log para cada nova conexão e o aumento do número de conexões ativas.



```
PS C:\Users\berna\OneDrive\Documentos\Computação\projetos\projeto-fccpd> & C:\Users\berna\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/berna/OneDrive/Documentos/Computação/projetos/projeto-fccpd/sistema-chat-distribuido/server.py
[ESCUTANDO] Servidor está escutando em 127.0.0.1:50000
[NOVA CONEXÃO] ('127.0.0.1', 63831) conectado.
[CONEXÕES ATIVAS] 1
[NOVA CONEXÃO] ('127.0.0.1', 63834) conectado.
[CONEXÕES ATIVAS] 2
[NOVA CONEXÃO] ('127.0.0.1', 63837) conectado.
[CONEXÕES ATIVAS] 3
```

**Figura 2 – Terminal do servidor exibindo o aceite de múltiplas conexões.**

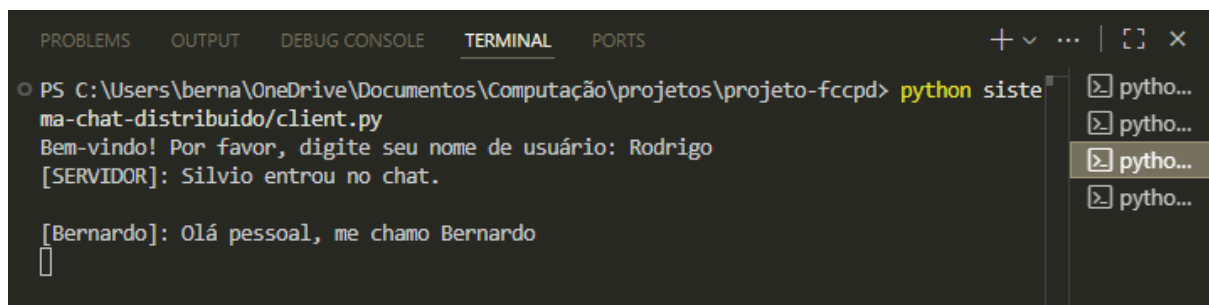
As Figuras 3, 4, 5, 6 e 7 mostram a interação do ponto de vista de três clientes: "Bernardo", "Rodrigo" e "Silvio". Nas Figuras 3, 4 e 5, o cliente "Bernardo" envia uma mensagem, que é recebida por todos os outros, como visto na Figura 6. Adicionalmente, quando o cliente "Silvio" se desconecta usando o comando sair, o servidor envia uma notificação para os clientes restantes, como também pode ser visto na Figura 7.



PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS + v ... | [ ] x

```
PS C:\Users\berna\OneDrive\Documentos\Computação\projetos\projeto-fccpd> & C:\Users\berna\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/berna/OneDrive/Documentos/Computação/projetos/projeto-fccpd/sistema-chat-distribuido/server.py
[ESCUTANDO] Servidor está escutando em 127.0.0.1:50000
[NOVA CONEXÃO] ('127.0.0.1', 63831) conectado.
[CONEXÕES ATIVAS] 1
[NOVA CONEXÃO] ('127.0.0.1', 63834) conectado.
[CONEXÕES ATIVAS] 2
[NOVA CONEXÃO] ('127.0.0.1', 63837) conectado.
[CONEXÕES ATIVAS] 3
Mensagem recebida de ('127.0.0.1', 63831): [Bernardo]: Olá pessoal, me chamo Bernardo

```

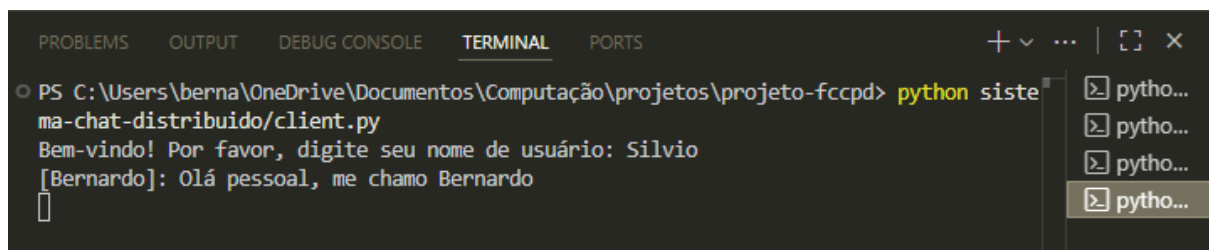


PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS + v ... | [ ] x

```
PS C:\Users\berna\OneDrive\Documentos\Computação\projetos\projeto-fccpd> python sistema-chat-distribuido/client.py
Bem-vindo! Por favor, digite seu nome de usuário: Rodrigo
[SERVIDOR]: Silvio entrou no chat.

[Bernardo]: Olá pessoal, me chamo Bernardo

```

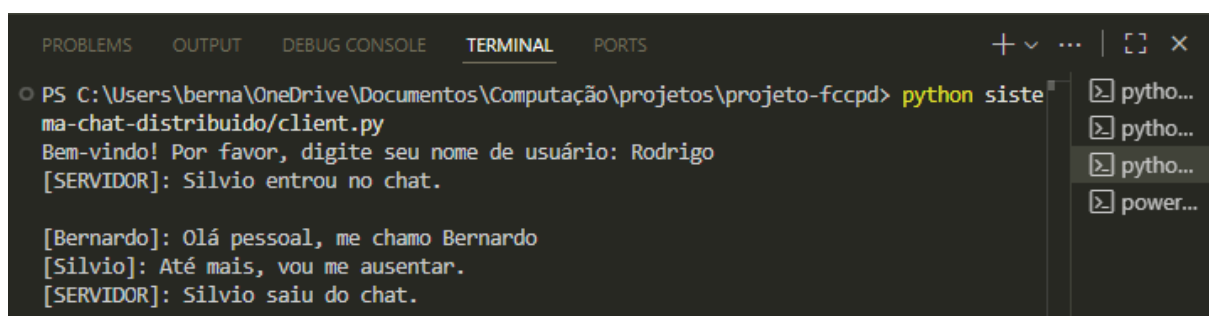


PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS + v ... | [ ] x

```
PS C:\Users\berna\OneDrive\Documentos\Computação\projetos\projeto-fccpd> python sistema-chat-distribuido/client.py
Bem-vindo! Por favor, digite seu nome de usuário: Silvio
[Bernardo]: Olá pessoal, me chamo Bernardo

```

Figuras 3, 4 e 5 – Cliente "Bernardo" enviando uma mensagem.



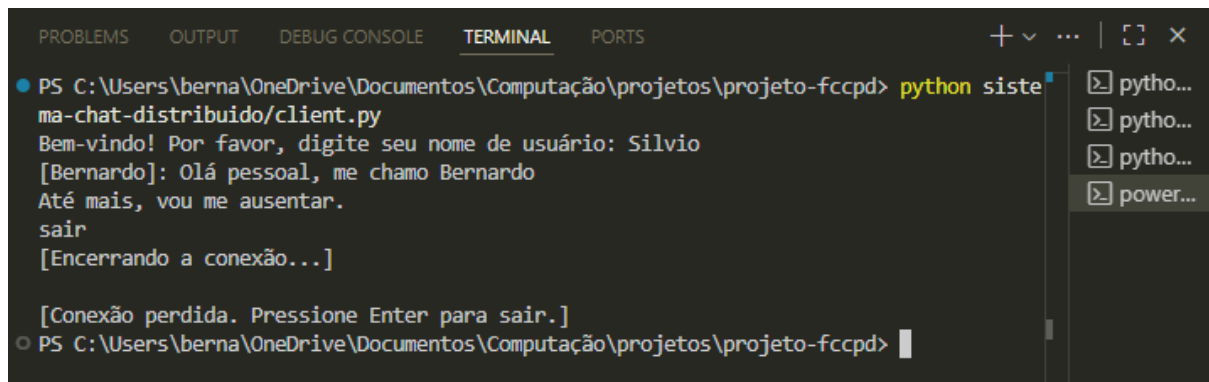
PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS + v ... | [ ] x

```
PS C:\Users\berna\OneDrive\Documentos\Computação\projetos\projeto-fccpd> python sistema-chat-distribuido/client.py
Bem-vindo! Por favor, digite seu nome de usuário: Rodrigo
[SERVIDOR]: Silvio entrou no chat.

[Bernardo]: Olá pessoal, me chamo Bernardo
[Silvio]: Até mais, vou me ausentar.
[SERVIDOR]: Silvio saiu do chat.

```

Figura 6 – Cliente "Rodrigo" recebendo a mensagem de "Bernardo" e a notificação de saída de "Silvio".



```
PS C:\Users\berna\OneDrive\Documentos\Computação\projetos\projeto-fccpd> python sistema-chat-distribuido/client.py
Bem-vindo! Por favor, digite seu nome de usuário: Silvio
[Bernardo]: Olá pessoal, me chamo Bernardo
Até mais, vou me ausentar.
sair
[Encerrando a conexão...]

[Conexão perdida. Pressione Enter para sair.]
PS C:\Users\berna\OneDrive\Documentos\Computação\projetos\projeto-fccpd>
```

**Figura 7 – Cliente "Silvio" se despedindo e usando o comando "sair" para uma desconexão limpa.**

Os testes validaram com sucesso todos os requisitos funcionais. O mecanismo de broadcast operou corretamente, e o sistema se mostrou robusto ao lidar tanto com desconexões limpas (via comando sair) quanto com desconexões abruptas (fechamento forçado do terminal do cliente).

## 4. CONCLUSÃO

O desenvolvimento deste projeto permitiu a aplicação prática e aprofundada dos conceitos teóricos de sistemas distribuídos, concorrência e sincronização. A implementação do sistema de chat na arquitetura cliente-servidor demonstrou ser uma abordagem eficaz para o problema proposto, e o uso de threads e locks em Python se provou uma ferramenta poderosa para gerenciar interações simultâneas de forma segura e eficiente.

O objetivo foi plenamente alcançado, resultando em uma aplicação funcional que não apenas resolve um problema prático de comunicação, mas também serve como um modelo claro de como estruturar uma aplicação concorrente e distribuída, tratando de questões essenciais como a prevenção de condições de corrida.

Como trabalhos futuros, o projeto poderia ser estendido para incluir funcionalidades mais avançadas, como a criação de salas de chat separadas, o envio de mensagens privadas entre usuários, a verificação de nomes de usuário únicos e a persistência do histórico de mensagens em um banco de dados ou arquivo.

