



UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA - CT

LARISSA FERNANDES DA SILVA

SERVIDOR DE CHAT COM SUPORTE A MÚLTIPLAS SALAS

VITÓRIA

2025

LARISSA FERNANDES DA SILVA

SERVIDOR DE CHAT COM SUPORTE A MÚLTIPLAS SALAS

Relatório apresentado à disciplina de Redes de Computadores, do curso de Engenharia de Computação como parte dos requisitos necessários à obtenção de nota.

Professor: Magnos Martinello

VITÓRIA
2025

SUMÁRIO

1. INTRODUÇÃO.....	2
2. IMPLEMENTAÇÃO	3
3. INTERFACE E FUNCIONAMENTO PRÁTICO.....	6
4. CONCLUSÃO.....	15
5. REFERÊNCIAS.....	16

1. INTRODUÇÃO

Os sistemas de chat têm passado por uma evolução significativa desde os primórdios da comunicação digital. Plataformas como o MSN Messenger revolucionaram a forma como as pessoas se conectavam online, permitindo trocas de mensagens instantâneas, chamadas de voz e vídeo, além do compartilhamento de arquivos. Essas ferramentas foram pioneiras em criar uma experiência de comunicação mais dinâmica e pessoal, aproximando pessoas de diferentes partes do mundo.

Deste modo, este projeto consiste na implementação de um servidor de chat que permite a criação e gerenciamento de múltiplas salas de chat, onde os clientes podem se conectar, interagir e trocar mensagens de forma organizada e eficiente. O sistema foi desenvolvido com o objetivo de proporcionar uma experiência de comunicação em tempo real, seguindo requisitos específicos para garantir a funcionalidade. O sistema permite que os clientes listem as salas disponíveis, entrem em uma sala específica e enviem mensagens que serão recebidas apenas pelos participantes da mesma sala. Para isso, foram implementados comandos que facilitam a navegação e a interação dos usuários.

Para a implementação, foi utilizada a linguagem de programação Python, conhecida por sua simplicidade e eficiência no desenvolvimento de aplicações de rede. As bibliotecas socket e threading foram empregadas para gerenciar a comunicação via sockets e a execução paralela de tarefas, respectivamente. A biblioteca socket permite a troca de dados entre cliente e servidor, enquanto o threading possibilita o gerenciamento de múltiplas conexões de forma assíncrona, garantindo que o servidor possa atender a diversos clientes simultaneamente. Logo, este relatório detalha as escolhas de implementação, as funcionalidades desenvolvidas, as instruções para execução do projeto e as possíveis melhorias futuras, proporcionando uma visão completa do sistema de chat implementado.

O código fonte do servidor e do cliente e o README.md estão disponíveis em:

<https://github.com/larissafernandesds/Servidor-de-Chat-com-Suporte-a-Multiplas-Salas>

2. IMPLEMENTAÇÃO

ESTRUTURA DO SERVIDOR

O servidor é responsável por escutar conexões de clientes e criar uma nova thread para cada cliente conectado.

- **Inicialização do Servidor**
 - O servidor é iniciado criando um socket TCP (`AF_INET`, `SOCK_STREAM`).
 - Ele é vinculado a um endereço IP e uma porta específica.
 - O servidor entra em modo de escuta (`listen()`) aguardando conexões de clientes.
- **Gerenciamento de Clientes**
 - Quando um cliente se conecta, o servidor aceita a conexão e cria uma nova thread para lidar com ele.
 - Cada cliente possui sua própria thread para evitar bloqueios no servidor.
 - A comunicação é feita através de mensagens enviadas pelo socket.
- **Tratamento de Mensagens**
 - O servidor recebe mensagens dos clientes e processa os comandos enviados.
 - Se um cliente se desconectar, a thread correspondente é encerrada.
 - O terminal do servidor exibe mensagens sobre a conexão e desconexão dos clientes, incluindo informações como apelido.

Escolhas de Implementação

- **Uso de Threads:**
 - Optou-se por `threading` para lidar com múltiplos clientes simultaneamente. Alternativamente, poderíamos usar `asyncio`, mas `threading` é mais intuitivo para o escopo deste projeto.
- **Protocolo TCP:**
 - TCP foi escolhido, ao invés de UDP, por fornecer uma conexão confiável e garantir que os pacotes cheguem na ordem correta.
- **Tratamento de Erros:**
 - Foram implementados blocos `try-except` para lidar com erros de conexão e interrupção abrupta de clientes.

- **Estrutura de Dados para Gerenciar Salas:**

- Há uma estrutura de dados (**dicionário**) para armazenar as salas e seus clientes, onde a chave é o nome da sala e o valor é um dicionário contendo a lista de clientes presentes. Isso facilita a criação, entrada e gerenciamento das salas.

- **Funções principais:**

- **start_server()**: Ela configura e inicializa o servidor, criando um socket TCP, vinculando-o ao endereço IP e porta definidos, e permitindo que ele aceite conexões simultâneas. O servidor entra em um loop onde, a cada nova conexão, uma nova thread é criada para lidar com a comunicação do cliente, chamando a função **handle_client**.
- **handle_client(client_socket, addr)**: Esta função é responsável por gerenciar a interação com cada cliente individualmente. Ela recebe, envia instruções e processa mensagens do cliente, realizando ações como listar salas, criar novas salas, entrar ou sair de salas, e enviar mensagens. Também lida com a desconexão do cliente, removendo-o da sala (se estiver em uma) e fechando a conexão quando o cliente sai do chat.

ESTRUTURA DO CLIENTE

O cliente é responsável por estabelecer uma conexão com o servidor e permitir que o usuário interaja com o chat.

- **Inicialização do Cliente**

- O cliente inicia criando um socket TCP (**AF_INET, SOCK_STREAM**). Ele se conecta ao servidor utilizando o endereço IP e a porta definidos pelo servidor. O cliente aguarda as instruções do servidor após a conexão ser estabelecida.

- **Interação com Servidor**

- Quando conectado ao servidor, o cliente solicita ao usuário que insira um apelido para identificação. O cliente então envia esse apelido para o servidor, que o registra e retorna uma mensagem de boas-vindas e as instruções.

- **Envio e Recebimento de Mensagens**

- Após a conexão inicial e envio do apelido, o cliente entra em um loop onde o usuário pode digitar mensagens, que são enviadas ao servidor. Se o usuário digitar `"/sair_chat"`, o cliente envia esse comando ao servidor e encerra a conexão. O socket do cliente é fechado, liberando recursos, e a aplicação imprime uma mensagem informando que o cliente foi desconectado.
- O cliente também recebe mensagens do servidor, como respostas a comandos ou mensagens de outros usuários na mesma sala. Além disso, o cliente pode receber mensagens informando sobre eventos, como a entrada ou saída de outros clientes das salas.

Escolhas de Implementação

- **Uso de Threads:**

- A implementação utiliza threads para lidar com a recepção de mensagens de forma assíncrona. Isso permite que o cliente continue interagindo com o servidor enquanto aguarda mensagens.

- **Protocolo TCP:**

- TCP foi escolhido, ao invés de UDP, por fornecer uma conexão confiável e garantir que os pacotes cheguem na ordem correta.

- **Tratamento de Erros:**

- O cliente implementa um tratamento básico de erros usando try-except para capturar falhas na recepção de mensagens. Em caso de desconexão, a função de recebimento de mensagens é encerrada, e a conexão é fechada.

- **Função principal:**

- `receive_messages(sock)`: é responsável por receber mensagens do servidor de forma assíncrona. Ela é executada em uma thread separada, garantindo que o cliente consiga receber mensagens enquanto o usuário envia novas mensagens.

3. INTERFACE E FUNCIONAMENTO PRÁTICO

No REAMDE.md do projeto, disponibilizado no repositório do Github, é mostrado como executar o programa e os requisitos necessários para isso. Nesta seção, serão demonstrados os comandos disponíveis, sendo que a interface do chat é a próprio terminal. Para testar o funcionamento do sistema, pode-se utilizar os comandos abaixo que estão disponíveis no chat:

- Comando */salas*: Lista todas as salas disponíveis no servidor. Esse comando é útil para ver as opções de salas criadas pelos usuários.
- Comando */criar <nome_da_sala>*: Cria uma nova sala de chat com o nome especificado. Se o nome da sala já existir, o sistema informará ao usuário.
- Comando */entrar <nome_da_sala>*: Entra em uma sala já existente. O sistema irá adicionar o usuário à sala especificada e permitirá que ele interaja com os outros clientes dessa sala.
- Comando */sair*: Sai da sala atual e não é mais possível receber mensagens dos clientes dessa antiga sala.
- Comando */sair_chat*: Desconecta o cliente do servidor de chat e encerra a aplicação do cliente. Este comando é usado para sair completamente do chat, encerrando a conexão com o servidor.
- Comando *Ctrl+C*: usado no terminal do servidor para interromper sua execução. Os demais comandos acima são usados no terminal do cliente.

Neste primeiro exemplo, vamos simular uma conexão da cliente “Lari” e logo após uma conexão de “Math”. Ambos entram na sala “REDES” criada por “Lari”. Depois da interação entre eles, ela sai do chat e ele se desconecta do servidor. É possível

observar no terminal do servidor as conexões sendo estabelecidas e a desconexão de “Math”.

Figura 1: Primeiro exemplo do servidor de chat, terminal de “Lari”.

```
larissa@larissa-15iil:~/T2$ python3 client.py
Digite seu apelido: Lari

__Conectado ao servidor__

Bem-vindo(a) ao chat, Lari! Use os comandos:
/salas - listar salas disponíveis
/criar <sala> - criar uma nova sala
/entrar <sala> - entrar em uma sala existente
/sair - sair da sala atual
/sair_chat - desconectar

/criar REDES
✅ Sala 'REDES' criada com sucesso!

/entrar REDES
✅ Você entrou na sala 'REDES'.

📝 Math entrou na sala 'REDES'.

A aula vai ser no labgrad?
[Math] sim!! Lab1...

/sair
♦ Você saiu da sala.
```

Figura 2: Terminal do usuário “Math”.

```

larissa@larissa-15iil:~/T2$ python3 client.py
Digite seu apelido: Math

__Conectado ao servidor__

Bem-vindo(a) ao chat, Math! Use os comandos:
/salas - listar salas disponíveis
/criar <sala> - criar uma nova sala
/entrar <sala> - entrar em uma sala existente
/sair - sair da sala atual
/sair_chat - desconectar

/salas
Salas disponíveis: REDES

/entrar REDES
✓ Você entrou na sala 'REDES'.

[Lari] A aula vai ser no labgrad?

sim!! Lab1...
📝 Lari saiu da sala 'REDES'.

/sair_chat

__Desconectado__

```

Figura 3: Terminal do servidor para o primeiro exemplo.

```

○ larissa@larissa-15iil:~/T2$ python3 server.py
Servidor rodando em 192.168.28.43:6789
Nova conexão de ('192.168.28.43', 58334) (aguardando apelido...)
Cliente conectado: Lari (('192.168.28.43', 58334))
Nova conexão de ('192.168.28.43', 49914) (aguardando apelido...)
Cliente conectado: Math (('192.168.28.43', 49914))
Math ('192.168.28.43', 49914) desconectou do chat.

```

No segundo exemplo, vamos simular uma conexão do cliente “Dipper” e logo após uma conexão de “Jade”, “Stephen” e “Fox Mulder”. É mostrado a criação das salas: “REDES” e “REDES2” e que a mudança e criação de salas pode ocorrer dentro do chat atual, sendo que se um cliente criar uma sala e entrar nela, para os outros usuários da sala atual é mostrado corretamente que ele saiu do chat atual. Dá para perceber também que o comando “/sair_chat” é case insensitive, ou seja, “/sair_CHAT” e “/SAIR_CHAT” também funcionam. Temos que algumas coisas não são permitidas, como:

- Entrar em uma sala que não existe;
- Criar sala com um nome já existente;
- Usar o comando “/sair” sem estar em alguma sala;
- Enviar mensagens sem estar em uma sala...

Figura 4: Terminal do servidor para o segundo exemplo.

```
larissa@larissa-15iil:~/T2$ python3 server.py
Servidor rodando em 192.168.28.43:6789
Nova conexão de ('192.168.28.43', 45706) (aguardando apelido...)
Cliente conectado: Dipper (('192.168.28.43', 45706))
Nova conexão de ('192.168.28.43', 57410) (aguardando apelido...)
Cliente conectado: Jade (('192.168.28.43', 57410))
Nova conexão de ('192.168.28.43', 55380) (aguardando apelido...)
Cliente conectado: Stephen (('192.168.28.43', 55380))
Nova conexão de ('192.168.28.43', 42546) (aguardando apelido...)
Cliente conectado: Fox Mulder (('192.168.28.43', 42546))
Dipper ('192.168.28.43', 45706) desconectou do chat.
Fox Mulder ('192.168.28.43', 42546) desconectou do chat.
Jade ('192.168.28.43', 57410) desconectou do chat.
Stephen ('192.168.28.43', 55380) desconectou do chat.
```

Figura 5: Terminal do usuário “Dipper”.

```
larissa@larissa-15i1l:~/T2$ python3 client.py
Digite seu apelido: Dipper

__Conectado ao servidor__

Bem-vindo(a) ao chat, Dipper! Use os comandos:
/salas - listar salas disponíveis
/criar <sala> - criar uma nova sala
/entrar <sala> - entrar em uma sala existente
/sair - sair da sala atual
/sair_chat - desconectar

/criar REDES
✅ Sala 'REDES' criada com sucesso!

/entrar REDES
✅ Você entrou na sala 'REDES'.

📝 Jade entrou na sala 'REDES'.

📝 Stephen entrou na sala 'REDES'.

[Stephen] Ola, pessoas!

📝 Stephen saiu da sala 'REDES'.

Oii, galera!
Hello!!
[Jade] Hiii

/sair
♦ Você saiu da sala.

/sair_chat

__Desconectado__
```

Figura 6: Terminal do usuário “Jade”.

```
larissa@larissa-15iil:~/T2$ python3 client.py
Digite seu apelido: Jade

__Conectado ao servidor__

Bem-vindo(a) ao chat, Jade! Use os comandos:
/salas - listar salas disponíveis
/criar <sala> - criar uma nova sala
/entrar <sala> - entrar em uma sala existente
/sair - sair da sala atual
/sair_chat - desconectar

/salas
Salas disponíveis: REDES

/entrar REDES
✓ Você entrou na sala 'REDES'.

📝 Stephen entrou na sala 'REDES'.

[Stephen] Ola, pessoas!

📝 Stephen saiu da sala 'REDES'.

[Dipper] Oii, galera!

[Dipper] Hello!!

Hiii
📝 Dipper saiu da sala 'REDES'.

/SAIR_CHAT

__Desconectado__
```

Figura 7: Terminal do usuário “Stephen”.

```
Digite seu apelido: Stephen

__Conectado ao servidor__

Bem-vindo(a) ao chat, Stephen! Use os comandos:
/salas - listar salas disponiveis
/criar <sala> - criar uma nova sala
/entrar <sala> - entrar em uma sala existente
/sair - sair da sala atual
/sair_chat - desconectar

/salas
Salas disponiveis: REDES

/entrar redes
❌ Sala não encontrada. Use /salas para ver as disponiveis ou /criar para criar uma nova.

/criar REDES
⚠️ Essa sala já existe! Tente outro nome.

/entrar REDES
✅ Você entrou na sala 'REDES'.

Ola, pessoas!
/salas
Salas disponiveis: REDES, REDES2

/entrar REDES2
✅ Você entrou na sala 'REDES2'.

Mudei de sala!
[Fox Mulder] Legal!

📄 Fox Mulder saiu da sala 'REDES2'.

/sair_CHAT

__Desconectado__
```

Figura 8: Terminal do usuário “Fox Mulder”.

```
larissa@larissa-15iil:~/T2$ python3 client.py
Digite seu apelido: Fox Mulder

__Conectado ao servidor__

Bem-vindo(a) ao chat, Fox Mulder! Use os comandos:
/salas - listar salas disponíveis
/criar <sala> - criar uma nova sala
/entrar <sala> - entrar em uma sala existente
/sair - sair da sala atual
/sair_chat - desconectar

/criar REDES2
✅ Sala 'REDES2' criada com sucesso!

/entrar REDES2
✅ Você entrou na sala 'REDES2'.

📝 Stephen entrou na sala 'REDES2'.

[Stephen] Mudei de sala!

Legal!
/sair_chat

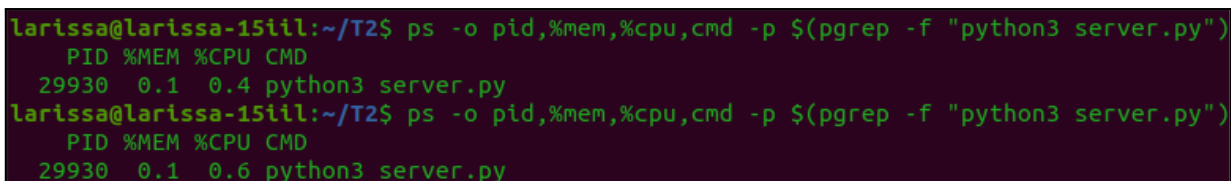
__Desconectado__
```

USO DA MEMÓRIA E ESCOLHA DE BACKLOG

Foi avaliada a escalabilidade do servidor, pois seria interessante descobrir quantas conexões simultâneas ele suporta sem problemas. O parâmetro backlog do `listen(backlog)` no servidor, define quantas conexões podem ficar pendentes antes que novas conexões sejam recusadas. Quando o servidor está ocupado e não pode aceitar novas conexões imediatamente, as conexões são colocadas em uma fila. Se a fila estiver cheia, novas conexões serão recusadas. Inicialmente o backlog foi definido com 30 e feito os testes.

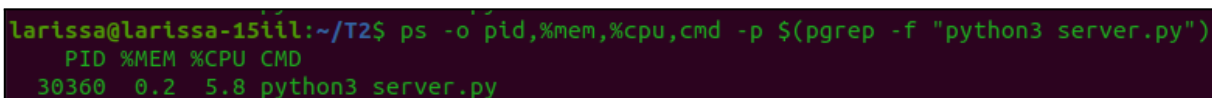
Para fazer esse teste, foi criado um novo arquivo chamado `scalability.py`, que simula várias conexões simultâneas e o envio de mensagens em uma sala do chat, depois ocorre a desconexão. É preciso executar o servidor e logo após o `scalability.py`. Com o comando `ps -o pid,%mem,%cpu,cmd -p $(pgrep -f "python3 server.py")` foi identificado a porcentagem de memória e cpu usados para sua respectiva quantidade de conexões (com 8GB de memória na máquina de teste). A seguir é variado o número de conexões simultâneas e o retorno do comando para avaliar o uso dos recursos.

Figura 9: Uso da memória com 100 conexões.



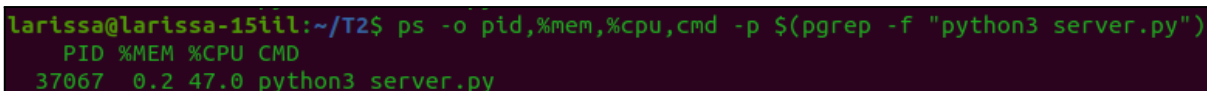
```
larissa@larissa-15iil:~/T2$ ps -o pid,%mem,%cpu,cmd -p $(pgrep -f "python3 server.py")
  PID %MEM %CPU CMD
  29930  0.1  0.4 python3 server.py
larissa@larissa-15iil:~/T2$ ps -o pid,%mem,%cpu,cmd -p $(pgrep -f "python3 server.py")
  PID %MEM %CPU CMD
  29930  0.1  0.6 python3 server.py
```

Figura 10: Uso da memória com 600 conexões.



```
larissa@larissa-15iil:~/T2$ ps -o pid,%mem,%cpu,cmd -p $(pgrep -f "python3 server.py")
  PID %MEM %CPU CMD
  30360  0.2  5.8 python3 server.py
```

Figura 11: Uso da memória com 1000 conexões, apresentando lentidão.



```
larissa@larissa-15iil:~/T2$ ps -o pid,%mem,%cpu,cmd -p $(pgrep -f "python3 server.py")
  PID %MEM %CPU CMD
  37067  0.2 47.0 python3 server.py
```

Para 2000 conexões, há problemas de broken pipe no servidor e o programa é abortado. Com 1500 essa situação também ocorre. Com 500 conexões tem um uso de 5% da CPU, com 30: há 0,2% e 20 tem 0,1%. Logo, é interessante um número de conexões igual ou abaixo de 600.

4. CONCLUSÃO

Temos que esse projeto permitiu a aplicação prática de conceitos fundamentais de programação em rede, como o uso de sockets e threads, além de explorar desafios de gerenciamento de conexões e comunicação entre clientes e servidor. A experiência adquirida com a criação de salas dinâmicas, o tratamento de mensagens e a implementação de comandos em Python foi enriquecedora e demonstrou como a teoria pode ser aplicada de forma prática e funcional.

Entre as melhorias futuras que podem ser implementadas, destacam-se: a implementação de uma Interface Gráfica, que tornaria a experiência do usuário mais amigável e acessível, especialmente para aqueles menos familiarizados com interfaces de linha de comando. Além disso, a inclusão de um Histórico de Mensagens permitiria que os usuários acessassem conversas anteriores, aumentando a utilidade do sistema. Por fim, a Autenticação de Usuários seria uma adição valiosa para garantir a identificação única de cada participante, evitando conflitos de apelidos e permitindo personalizações como perfis e status personalizados.

5. REFERÊNCIAS

GEEKSFORGEEKS. Multithreading in Python | Set 1. Disponível em: <<https://www.geeksforgeeks.org/multithreading-python-set-1/>>.

HOWTO sobre a Programação de Soquetes — documentação Python 3.8.9. Disponível em: <<https://docs.python.org/pt-br/3.8/howto/sockets.html>>.

PROGRAMMINGKNOWLEDGE. Python Socket Programming Tutorial 7 - TCP/IP Client and Server. Disponível em: <https://www.youtube.com/watch?v=BIQbUV_W954>. Acesso em: 06 fev. 2025.

VERMA, K. Socket Programming in Python - GeeksforGeeks. Disponível em: <<https://www.geeksforgeeks.org/socket-programming-python/>>.