

UNIVERSIDADE FEDERAL DE ALAGOAS
CIÊNCIA DA COMPUTAÇÃO

DANIEL DE ARAUJO ALBUQUERQUE
LAURA BEATRIZ LINS RAMOS MAINERO
PEDRO CABRAL GOMES
YURI RAPHAEL MOTA DE ARAÚJO BARBOSA

PROJETO DE REDES DE COMPUTADORES
Sistema de Fila de Atendimento

MACEIÓ

2024

DANIEL DE ARAUJO ALBUQUERQUE
LAURA BEATRIZ LINS RAMOS MAINERO
PEDRO CABRAL GOMES
YURI RAPHAEL MOTA DE ARAÚJO BARBOSA

PROJETO DE REDES DE COMPUTADORES

Sistema de Fila de Atendimento

Trabalho apresentado na matéria de Redes de Computadores do curso de Engenharia e Ciência da Computação da Universidade Federal de Alagoas.

Prof. Almir Pereira Guimarães

MACEIÓ

2024

1. Principais Funcionalidades da Aplicação.....	4
2. Protocolos Implementados.....	5
3. Dificuldades Encontradas.....	5
4. O que Poderia ter sido Implementado a Mais.....	6
5. Conclusão.....	7
6. Código Fonte.....	7
• Codigo de servidor.py.....	7
• Codigo de cliente.py.....	13
• Codigo de atendente.py.....	16

1. Principais Funcionalidades da Aplicação

O projeto implementa um sistema de "Fila de Atendimento" (Help Desk) em modo texto, onde múltiplos "Clientes" podem se conectar e entrar numa fila para serem atendidos por múltiplos "Atendentes".

- **Conexão e Identificação:** O sistema distingue dois tipos de usuários (`cliente.py` e `atendente.py`). Ao conectar, cada usuário se identifica ao servidor como "CLIENTE" ou "ATENDENTE" e envia seu nome.
- **Gerenciamento de Fila (Lado Servidor):** O servidor (`servidor.py`) mantém duas listas separadas: `fila_clientes` e `fila_atendentes`. A gestão de concorrência a essas listas é controlada através de `threading.Lock()` para evitar "race conditions".
- **Matchmaking (Formação de Pares):** Quando há pelo menos um cliente e um atendente disponíveis em suas respectivas filas, o servidor automaticamente "retira" o primeiro de cada fila e estabelece uma conexão de chat privada entre eles.
- **Comunicação em Tempo Real:** Uma vez conectados, o cliente e o atendente podem trocar mensagens em tempo real. O servidor atua como um "intermediário" (relay), repassando as mensagens de um para o outro através da função `repassar_chat`.
- **Feedback ao Usuário (UX):** A aplicação fornece feedback constante ao usuário.
 - **Clientes** recebem sua posição na fila (ex: "Você é o número 2 na fila...").
 - **Ambos** são notificados quando a conexão é estabelecida (ex: "Conexão estabelecida! Você está falando com [Nome]").
 - **Ambos** são notificados quando o outro lado desconecta (ex: "Atendimento encerrado. O cliente [Nome] desconectou.").
- **Gerenciamento de Desconexão:** Se um usuário encerra a conexão, o servidor detecta isso e envia uma mensagem de protocolo `FIM` para o parceiro. Se um Cliente desconecta, o Atendente é realocado para a fila de espera. Se um Atendente desconecta, o Cliente é encerrado.

2. Protocolos Implementados

Para a comunicação de *sockets* TCP, nós desenvolvemos um protocolo simples de camada de aplicação baseado em texto para o controle de estado e repasse de mensagens.

- **Protocolo de Conexão (Handshake):**
 - Usuário (Cliente ou Atendente) estabelece uma conexão socket com o Servidor.
 - Usuário envia sua **TIPO** (string "CLIENTE" ou "ATENDENTE").
 - Usuário envia seu **NOME** (string "Yuri").
- **Protocolo de Comunicação (Servidor -> Cliente/Atendente):** Para controlar o estado da aplicação cliente, o servidor envia mensagens prefixadas que são "comandos". O programa cliente lê esses comandos, mas **não os exibe** para o usuário final.
 - **FILA:< posição >**: Indica uma mensagem de status da fila.
 - **CONECTADO:< nome >**: Sinaliza que o par foi formado e inicia o chat
 - **FIM:< mensagem >**: Notifica o encerramento do chat e o motivo.
- **Protocolo de Comunicação (Cliente/Atendente -> Servidor):**
 - Qualquer mensagem enviada sem um prefixo é tratada como uma mensagem de chat.
 - O programa cliente formata a mensagem antes de enviar (ex: **[Yuri]: Olá!**).
 - O servidor recebe essa mensagem e, na função **repassar_chat**, a retransmite "na íntegra" para o destino.

3. Dificuldades Encontradas

- **Gerenciamento de Concorrência (Threads):** A maior dificuldade foi garantir que as threads do servidor não causassem conflitos ao acessar as **fila_clientes** e **fila_atendentes** ao mesmo tempo.
 - **Solução:** Implementamos um **threading.Lock (lock_das_fila)**, que é usado em todas as operações de adição (**append**) ou remoção (**pop**) das listas, garantindo que apenas uma thread possa modificá-las por vez.
- **"Race Condition" no Terminal do Cliente:** Tivemos um problema significativo de usabilidade onde o **input()** da thread principal e o **print()** da thread de escuta (**ouvir_servidor**) "brigavam" pelo controle do terminal.
 - Isso causava prompts **Sua mensagem**: duplicados ou mensagens recebidas que quebravam a linha que o usuário estava digitando.

- **Solução:** Centralizamos o controle. A thread principal (`main`) foi modificada para **nunca** imprimir o prompt, ela apenas executa `msg = input()`. A thread `ouvir_servidor` tornou-se a única responsável por imprimir *tanto* as mensagens recebidas *quanto* o prompt `Sua mensagem:` subsequente, garantindo uma interface limpa.
- **Gerenciamento de Estado Pós-Chat:** Um bug encontrado foi que o Atendente, após o fim de um chat, continuava preso num loop que imprimia "Aguardando conexão..." a cada nova mensagem.
 - **Solução:** Implementamos uma verificação `if socket.fileno() == -1`: na thread principal. Quando a thread `ouvir_servidor` recebe a mensagem `FIM`: e fecha o socket, a thread principal detecta isso na próxima tentativa de `input()` e encerra o loop `while True`:, finalizando o programa corretamente.

4. O que Poderia ter sido Implementado a Mais

- **Protocolo Robusto (JSON):** Embora nosso protocolo de texto seja funcional, uma migração para o formato JSON para a troca de mensagens (ex: `{"comando": "conectado", "nome": "Yuri"}`) tornaria a comunicação mais fácil de estender, validar e manter.
- **Múltiplos Departamentos:** Implementar uma funcionalidade que permita ao Cliente escolher um "departamento" (ex: "Vendas", "Suporte"), e o servidor o direcionasse para uma fila específica para aquele setor.
- **Log de Conversas:** O servidor poderia registrar o conteúdo das conversas em um arquivo de log (ex: CSV ou banco de dados) para fins de auditoria ou histórico.

5. Conclusão

O projeto cumpriu todos os requisitos obrigatórios da disciplina. Foi desenvolvida uma aplicação cliente-servidor funcional em Python, utilizando sockets para comunicação de baixo nível e threads para o gerenciamento de múltiplas conexões simultâneas (concorrência). O desenvolvimento permitiu aplicar na prática os conceitos de protocolos de camada de aplicação, gerenciamento de estado e os desafios da programação concorrente.

6. Código Fonte

- **Código de servidor.py**

```
import socket
import threading

user_states = {}
fila_clientes = []
fila_atendentes = []
ultima_posicao = {}
global_lock = threading.Lock()

def tentar_formar_par():
    global_lock.acquire()

    fila_clientes[:] = [c for c in fila_clientes if c in
user_states]
    fila_atendentes[:] = [a for a in fila_atendentes if a
in user_states]

    if fila_clientes and fila_atendentes:
        print("[MATCHMAKER] Formando um par!")
        cliente_conn = fila_clientes.pop(0)
        atendente_conn = fila_atendentes.pop(0)

        cliente_name = user_states[cliente_conn]['name']
        atendente_name =
user_states[atendente_conn]['name']

        user_states[cliente_conn]['state'] = 'chat'
        user_states[cliente_conn]['partner'] =
atendente_conn

        user_states[atendente_conn]['state'] = 'chat'
        user_states[atendente_conn]['partner'] =
```

```

cliente_conn

try:
    cliente_conn.send(f"CONECTADO: Você está
falando com {atendente_name}.".encode())
except Exception:
    pass

try:
    atendente_conn.send(f"CONECTADO: Você está
falando com {cliente_name}.".encode())
except Exception:
    pass

else:
    for i, conn in enumerate(fila_clientes):
        posicao_atual = i + 1
        ultima = ultima_posicao.get(conn)

        if ultima != posicao_atual:
            try:
                conn.send(f"FILE:{posicao_atual}".encode())
                ultima_posicao[conn] = posicao_atual
            except Exception:
                pass

    for conn in fila_atendentes:
        try:
            conn.send("FILE:OK".encode())
        except Exception:
            pass

```

```

global_lock.release()

def handle_disconnect(conn):
    global_lock.acquire()

    if conn not in user_states:
        global_lock.release()
        return

    disconnected_user = user_states.pop(conn)
    print(f"[DISCONNECT] {disconnected_user['name']} "
          f"({disconnected_user['type']}) desconectou.")

    if conn in fila_clientes:
        fila_clientes.remove(conn)
    if conn in fila_atendentes:
        fila_atendentes.remove(conn)

    if conn in ultima_posicao:
        ultima_posicao.pop(conn)

    partner_conn = disconnected_user['partner']
    partner_to_close = None

    if partner_conn and partner_conn in user_states:
        partner_state = user_states[partner_conn]
        print(f"[DISCONNECT] Parceiro "
              f"{partner_state['name']} será notificado.")

    try:
        partner_conn.send(f"FIM:{disconnected_user['name']} "
                          "desconectou.".encode())

```

```

        except Exception:
            pass

        if disconnected_user['type'] == 'CLIENTE':
            print(f"[RE-QUEUE] {partner_state['name']} ")
            (Atendente) de volta à fila.")
            partner_state['state'] = 'queue'
            partner_state['partner'] = None
            fila_atendentes.append(partner_conn)

        elif disconnected_user['type'] == 'ATENDENTE':
            print(f"[DISCONNECT] {partner_state['name']} ")
            (Cliente) será desconectado.")
            user_states.pop(partner_conn)
            if partner_conn in fila_clientes:
                fila_clientes.remove(partner_conn)
                partner_to_close = partner_conn

    global_lock.release()

    try:
        conn.close()
    except Exception:
        pass

    if partner_to_close:
        try:
            partner_to_close.close()
        except Exception:
            pass

tentar_formar_par()

```

```

def handle_connection(conn, addr):
    try:
        tipo = conn.recv(1024).decode()
        if tipo not in ["CLIENTE", "ATENDENTE"]:
            raise Exception("Tipo inválido")

        conn.send("NOME:".encode())
        nome = conn.recv(1024).decode()
        if not nome:
            raise Exception("Nome vazio")
    except Exception as e:
        print(f"[{addr}] falhou na identificação: {e}")
        conn.close()
        return

    print(f"[CONNECT] {nome} ({tipo}) conectou de {addr}.")
```

global_lock.acquire()

user_states[conn] = {'name': nome, 'type': tipo, 'state': 'queue', 'partner': None}

if tipo == 'CLIENTE':
 fila_clientes.append(conn)
else:
 fila_atendentes.append(conn)

global_lock.release()

tentar_formar_par()

while True:
 try:
 dados = conn.recv(1024)
 if not dados:
 break

```

        global_lock.acquire()
        if conn not in user_states:
            global_lock.release()
            break

        current_state = user_states[conn]
        if current_state['state'] == 'chat':
            partner_conn = current_state['partner']
            if partner_conn and partner_conn in
user_states:
                try:

                    partner_conn.send(f"[{current_state['name']}]")
                    {dados.decode()}.encode())
                except Exception:
                    pass
                global_lock.release()

            except Exception:
                break
        handle_disconnect(conn)

servidor_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
servidor_socket.bind(('0.0.0.0', 12345))
servidor_socket.listen(5)
print("Servidor (v.Fila Inteligente) ouvindo na porta
12345...")

while True:
    conn, addr = servidor_socket.accept()
    threading.Thread(target=handle_connection, args=(conn,

```

```
addr)).start()
```

- **Código de cliente.py**

```
import socket
import threading

def ouvir_servidor(sock):
    print("\nSua mensagem: ", end="", flush=True)

    while True:
        try:
            dados = sock.recv(1024).decode()
            if not dados:
                print("\n*** Desconectado do servidor.\n***")
                break

            if dados.startswith("FILA:"):
                posicao = dados.split(":", 1)[1]
                print(f"\r{' ' * 70}\r(Você é o número {posicao} na fila. Por favor, aguarde...)\nSua mensagem: ", end="", flush=True)

            elif dados.startswith("CONECTADO:"):
                mensagem_limpaa = dados.split("CONECTADO:")[1]
                print(f"\r{' ' * 70}\r*** Conectado!\n{mensagem_limpaa} ***\nSua mensagem: ", end="", flush=True)

            elif dados.startswith("FIM:"):
                mensagem_limpaa = dados.split("FIM:")[1]
                print(f"\r{' ' * 70}\r*** {mensagem_limpaa}
```

```

***\n*** Sessão encerrada. ***\n", end="", flush=True)
break

else:
    print(f"\r{' ' * 70}\r{dados}\nSua
mensagem: ", end="", flush=True)

except:
    break

print("Pressione Enter para sair.")
sock.close()

cliente_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
try:
    cliente_socket.connect(('127.0.0.1', 12345))
except ConnectionRefusedError:
    print("Não foi possível conectar ao servidor. Verifique
se ele está online.")
    exit()

cliente_socket.send("CLIENTE".encode())

try:
    dados_servidor = cliente_socket.recv(1024).decode()
    if dados_servidor == "NOME:":
        meu_nome = input("Digite seu nome: ")
        cliente_socket.send(meu_nome.encode())
        print("\nProcurando um atendente...")
    else:
        print(f"Erro de protocolo inesperado:
{dados_servidor}")

```

```

        cliente_socket.close()
        exit()

except Exception as e:
    print(f"Erro ao enviar nome: {e}")
    cliente_socket.close()
    exit()

threading.Thread(target=ouvir_servidor,
                  args=(cliente_socket,)).start()

try:
    while True:
        mensagem = input()
        if cliente_socket.fileno() != -1:
            cliente_socket.send(mensagem.encode())
        else:
            break
except:
    print("Saindo...")

cliente_socket.close()

```

- **Código de atendente.py**

```

import socket
import threading

def ouvir_servidor(sock):
    print("\nSua mensagem: ", end="", flush=True)
    while True:
        try:
            dados = sock.recv(1024).decode()

```

```

        if not dados:
            print(f"\r{' ' * 70}\r\n*** DESCONECTADO do
servidor. ***")
            break

        if dados.startswith("FILA:"):
            print(f"\r{' ' * 70}\r(Aguardando
clientes...)\nSua mensagem: ", end="", flush=True)

        elif dados.startswith("CONECTADO:"):
            mensagem_limpaa = dados.split("CONECTADO:
") [1]
            print(f"\r{' ' * 70}\r*** Cliente
conectado! {mensagem_limpaa} ***\nSua mensagem: ", end="",
flush=True)

        elif dados.startswith("FIM:"):
            mensagem_limpaa = dados.split("FIM:") [1]
            print(f"\r{' ' * 70}\r*** {mensagem_limpaa}
***\n*** Chat encerrado. ***\nSua mensagem: ", end="",
flush=True)

        else:
            print(f"\r{' ' * 70}\r{dados}\nSua
mensagem: ", end="", flush=True)
        except:
            print(f"\r{' ' * 70}\r\n*** Erro de conexão ou
socket fechado. ***")
            break

        print("Encerrando thread de escuta...")

```

try:

```

atendente_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
atendente_socket.connect(('127.0.0.1', 12345))
except ConnectionRefusedError:
    print("Servidor não encontrado. Encerrando.")
    exit()

try:
    atendente_socket.send("ATENDENTE".encode())

    dados_servidor = atendente_socket.recv(1024).decode()
    if dados_servidor == "NOME:":
        meu_nome = input("Digite seu nome de atendente: ")
        atendente_socket.send(meu_nome.encode())
        print("\nVocê está online e aguardando clientes.")
    else:
        raise Exception("Protocolo quebrado")

    thread_escuta = threading.Thread(target=ouvir_servidor,
args=(atendente_socket,))
    thread_escuta.start()

    while thread_escuta.is_alive():
        mensagem = input()
        if atendente_socket.fileno() == -1 or not
thread_escuta.is_alive():
            break
        atendente_socket.send(mensagem.encode())

except KeyboardInterrupt:
    print("\nSaindo... (Ctrl+C pressionado). Fechando
conexão.")
except Exception as e:

```

```
print(f"\nErro no loop principal: {e}")  
  
finally:  
    print("Limpando e encerrando conexão...")  
    if atendente_socket.fileno() != -1:  
        atendente_socket.close()  
    thread_escuta.join()  
    print("Aplicação encerrada.")
```