Relatório — Etapa 3 (v3-final)

Projeto: Servidor de Chat Multiusuário (Tema A) — *Etapa 3: Sistema completo*
Aluno: Lucas Henrique Vieira da Silva
Data: 06/10/2025

Resumo

A terceira etapa do projeto teve como objetivo concluir o desenvolvimento do **servidor de chat concorrente** iniciado nas etapas anteriores. Nesta fase, foram integrados todos os componentes do sistema, com foco na **sincronização entre threads**, na **organização das mensagens por meio de uma fila thread-safe** e na **análise crítica dos mecanismos de concorrência**.

O resultado é um servidor capaz de lidar com múltiplos clientes simultaneamente, garantindo o envio ordenado de mensagens, o registro de logs de forma segura e o encerramento controlado das threads e recursos.

Desenvolvimento

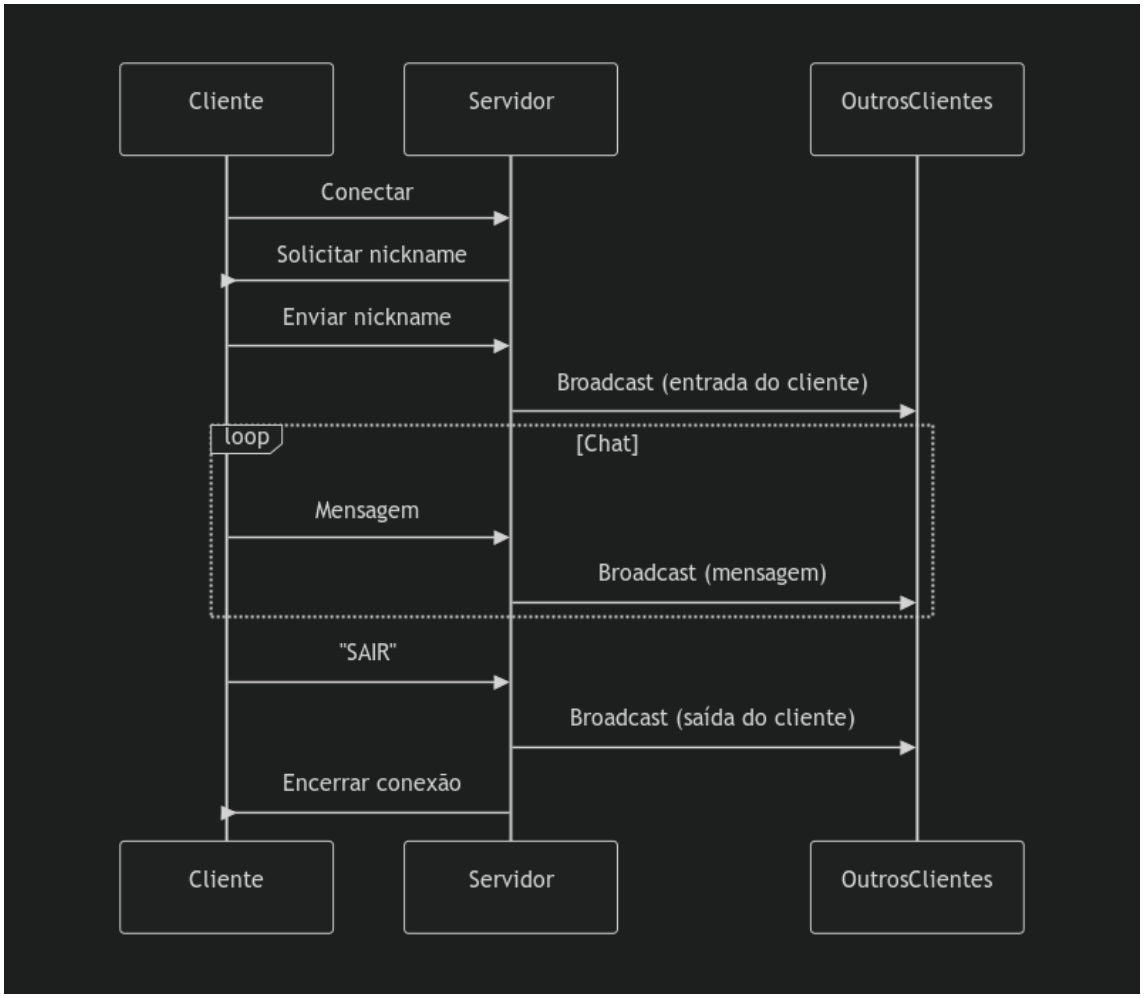
A principal adição nesta etapa foi a implementação de uma **fila de mensagens segura para threads**, desenvolvida nos arquivos thread_safe_queue.c e thread_safe_queue.h. Essa estrutura utiliza mutexes e semáforos para coordenar o acesso entre as threads que produzem mensagens (clientes) e a thread consumidora (responsável por despachá-las).

O servidor, implementado em servidor_chat.c, passou a utilizar uma **thread trabalhadora** dedicada ao processamento das mensagens, retirando-as da fila e enviando-as aos demais clientes conectados. Essa separação melhorou a organização do código e reduziu a possibilidade de conflitos entre as threads individuais.

As estruturas compartilhadas — como a lista de clientes e o histórico de mensagens — receberam **proteção adicional com mutexes**. O histórico é atualizado a cada nova mensagem e enviado automaticamente a novos clientes, tudo sob controle de sincronização para evitar condições de corrida.

Diagrama de sequência

O diagrama abaixo representa o comportamento do projeto, omitindo alguns funcionamentos internos do servidor, como a fila de mensagens.



Problemas Identificados e Correções

Problema	**Sugestão da IA**	**Correção Implementada**
Race condition na remoção de clientes	Usar mutex ao acessar a lista de clientes	✅ Mutex mutex_clientes adicionado
Deadlock no encerramento da thread trabalhadora	Sinalização com semáforo	✅ Uso de sem_post na fila
Envio parcial com send()	Verificar retorno de send()	✅ Loop de envio até completar
Bloqueio prolongado de mutex do histórico	Liberar mutex antes de operações de I/O	✅ Seção crítica reduzida

Resultados

Foram realizados testes com múltiplos clientes conectados simultaneamente, tanto automatizados quanto interativos. O servidor se mostrou capaz de transmitir todas as mensagens corretamente, mantendo a consistência do histórico e registrando eventos de forma adequada através da biblioteca libtslog.

Conclusão

Com a Etapa 3, o servidor de chat foi concluído e consolidado como um sistema **completo, funcional e seguro**. As melhorias introduzidas nesta fase reforçaram a estabilidade e a escalabilidade do projeto, garantindo o correto gerenciamento das threads e dos recursos compartilhados.

Mapeamento

Requisito	Arquivo	Implementação	Status
Threads	servidor_chat.c	pthread_create() para clientes e worker	✅
Exclusão Mútua	servidor_chat.c, libtslog.c, thread_safe_queue.c	pthread_mutex_t em múltiplos recursos	✅
Semáforos	thread_safe_queue.c	sem_t empty, full para sincronização da fila	✅
Sockets	servidor_chat.c, cliente_chat.c	socket(), bind(), listen(), accept()	✅
Logging Concorrente	libtslog.c	log_write() com mutex	✅
Fila Thread-Safe	thread_safe_queue.c	queue_put(), queue_get() com semáforos	✅
Broadcast	servidor_chat.c	broadcast_message(), broadcast_message_sync()	✅
Histórico de Mensagens	servidor_chat.c	add_to_history(), send_history()	✅
Graceful Shutdown	**servidor_chat.c**	**signal_handler(), server_running flag**	✅

Análise Crítica com IA

Com o auxílio de uma análise automatizada realizada por uma ferramenta de IA, foram revisados os aspectos de concorrência e sincronização do código.

Prompt Utilizado

"*Analise o seguinte código C para um servidor de chat multithread. Identifique possíveis race conditions, deadlocks, vazamentos de memória e problemas de sincronização. Dê sugestões de correção e realize uma análise técnica***."**

Entre os principais pontos levantados estão:

- o risco de bloqueio da thread de mensagens durante o encerramento;
- a possibilidade de uso indevido de memória caso um cliente fosse removido enquanto ainda estivesse sendo acessado por outra thread;
- a ausência de tratamento para envios parciais de dados via send();
- e a necessidade de otimização no tempo de bloqueio dos mutexes usados no histórico.

Algumas das medidas corretivas adotadas eliminaram alguns dos riscos de bloqueio e inconsistência, tornando o servidor mais estável. Além disso, a separação clara entre as threads produtoras e a thread consumidora de mensagens simplificou a lógica de sincronização, reduzindo o acoplamento entre as partes do sistema e facilitando a manutenção do código.