Trabalho Final
Programação Concorrente (C / C++)

Objetivo geral

Desenvolver um **sistema concorrente em rede** completo (cliente/servidor TCP ou UDP), que demonstre domínio prático dos conceitos de programação concorrente estudados no curso: **threads, exclusão mútua, semáforos, variáveis de condição, monitores e sockets**.

O projeto deve ser entregue em **3 etapas** (arquitetura + logging thread-safe; protótipo CLI com comunicação básica; sistema completo com cliente/servidor funcionando).

Além disso, os alunos deverão incluir um **relatório de análise crítica com auxílio de IA (LLMs)**, documentando potenciais problemas de concorrência (race conditions, deadlocks, starvation) e como foram mitigados.

Temas

O aluno escolhe **1 tema dentre os 3 abaixo** ou pode propor um tema próprio, desde que aprovado pelo professor. Todos devem implementar **comunicação via sockets**.

Tema A — Servidor de Chat Multiusuário (TCP)

- **Obrigatórios**
    - Servidor TCP concorrente aceitando múltiplos clientes.
    - Cada cliente atendido por thread; mensagens retransmitidas para os demais (broadcast).
    - Logging concorrente de mensagens (usando `libtslog`)
    - Cliente CLI: conectar, enviar/receber mensagens
    - Proteção de estruturas compartilhadas (lista de clientes, histórico).
- **Opcionais**
    - Autenticação simples (senha); mensagens privadas; criptografia (TLS); filtros de palavras.

Tema B — Mini Servidor Web (HTTP)

- **Obrigatórios**
    - Servidor HTTP/1.0 concorrente (thread por conexão ou thread-pool).
    - Atender requisições GET, servindo arquivos de um diretório.
    - Logging concorrente de requisições (usando `libtslog`)
    - Fila de conexões pendentes com limite configurável.
    - CLI para iniciar servidor, parar, mostrar estatísticas.
- **Opcionais**
    - Suporte a keep-alive; cache em memória; execução de script python.

Tema c — Agendador Distribuído de Tarefas

- **Obrigatórios**
    - Servidor central que mantém fila de jobs (scripts python ou lua, por exemplo).
    - Clientes (workers) conectam-se, solicitam jobs, executam e retornam resultados.
    - Logging concorrente de solicitações (usando `libtslog`)
    - Jobs possuem prioridade e timeout.
    - Monitor CLI no servidor: listar jobs, workers ativos, estatísticas.
- **Opcionais**
    - Persistência em SQLite; jobs dependentes; re-dispatch em caso de falha de worker.

Requisitos gerais (para todos os temas)

1. **Threads**: uso de `std::thread` ou pthreads para concorrer clientes/conexões.
2. **Exclusão mútua**.
3. **Semáforos e condvars**: controle de filas, slots, sincronização.
4. **Monitores**: encapsular sincronização em classes (ex.: `ThreadSafeQueue`).
5. **Sockets**.
6. **Gerenciamento de recursos**
7. **Tratamento de erros**: exceções e mensagens amigáveis no CLI.
8. **Logging concorrente**: uso obrigatório da biblioteca `libtslog` (ver Etapa 1).
9. **Documentação**: diagramas de sequência (cliente-servidor).
10. **Build**: Makefile ou CMakeLists.txt funcional em Linux (ou Windows).
11. **Uso de IA/LLMs**: relatório de análise crítica (detecção de possíveis race conditions, deadlocks, etc., com prompts e resumo de sugestões).

Etapas de Entrega

Cada etapa é um *commit* ou *tag* identificado no repositório (github ou gitlab) e acompanhado por um release/issue com instruções (repo Git com tags: `v1-logging`, `v2-cli`, `v3-final`). Entregue também um PDF com relatório técnico quando solicitado na etapa. O commit deve ser avisado e a URL no repositório do github deve ser informada por e-mail para bidu @ ci.ufpb.br , com o título: **"[LPII-251-E003-1/2/3] NOME - MATRICULA"** (onde 1/2/3 refere-se à etapa relacionada à mensagem).

Etapa 1 — Biblioteca `libtslog` + Arquitetura (4 pontos)

- Implementar biblioteca de logging **thread-safe** com API clara.
- Teste CLI simulando múltiplas threads gravando logs.
- Arquitetura inicial do projeto (headers principais + diagramas).
- Tag no repositório: `v1-logging`.
- Entrega: até 26/09/2025 23h59

Etapa 2 — Protótipo CLI de Comunicação (4 pontos)

- Implementar cliente/servidor mínimo **funcionando em rede**.
    - Ex.: chat: cliente conecta e envia mensagens; servidor retransmite.
    - Web: servidor responde a GET básico.
    - Agendador: cliente envia job e servidor reconhece.
- Logging integrado (`libtslog`).
- Scripts de teste (simulação de múltiplos clientes).
- Tag: `v2-cli`.
- Entrega: até 03/10/2025 23h59

Etapa 3 — Sistema completo (4 pontos)

- Funcionalidades obrigatórias do tema concluídas.
- Logging integrado.
- Relatório final:
    - Diagrama de sequência cliente-servidor.
    - Mapeamento requisitos → código.
    - Relatório de análise com IA.
- Opcional: vídeo ≤ 3 min demonstrando execução (link ou embed no README.md).
- Tag: `v3-final`.
- Entrega: até 06/10/2025 12h00

Avaliação

- **Funcionalidade (40%)**: requisitos obrigatórios atendidos.
- **Qualidade do código (20%)**: modularidade, smart pointers, RAII.
- **Concorrência e robustez (20%)**: ausência de race/deadlocks, uso correto de sincronização.
- **Documentação, testes e configuração (10%)**: Diagramas, README, build
- **IA / análise crítica (10%)**: relatório de uso de LLMs e valor agregado

---