UFPB
UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA

Linguagem de Programação II
Mini Web Server (MWS)
Etapa 3

Tobias Freire Numeriano

RELATÓRIO DE ANÁLISE CRÍTICA

1. Introdução

O sistema implementa um modelo cliente-servidor em C/C++, com funcionalidades concorrentes e logging integrado. A análise a seguir identifica possíveis problemas de concorrência e sugere estratégias de mitigação.

---

2. Potenciais Problemas de Concorrência

2.1. Race Conditions

Descrição:
Race conditions ocorrem quando múltiplas threads/processos acessam e modificam dados compartilhados simultaneamente, levando a resultados imprevisíveis.

Locais Potenciais no Código:

- Manipulação de conexões em server.cpp e conn.cpp.
- Escrita em arquivos de log via libtslog.cpp.

Exemplo:
Se múltiplos clientes acessam recursos compartilhados (ex: lista de conexões, arquivos de log) sem sincronização adequada, pode haver corrupção de dados.

Mitigação:

- Uso de mutexes (ex: pthread_mutex_t) para proteger regiões críticas.
- Garantir que funções de logging sejam thread-safe.
- Revisar se todas as variáveis globais ou estáticas acessadas por múltiplas threads estão protegidas.

2.2. Deadlocks

Descrição:
Deadlocks ocorrem quando duas ou mais threads esperam indefinidamente por recursos bloqueados entre si.

Locais Potenciais no Código:

- Se múltiplos mutexes forem usados em diferentes ordens em conn.cpp ou libtslog.cpp.
- Operações de leitura/escrita simultâneas em arquivos de log.

Mitigação:

- Sempre adquirir múltiplos locks na mesma ordem.

- Minimizar o tempo de bloqueio de mutexes.
- Utilizar timeout em locks, se possível.

2.3. Starvation

Descrição:
Starvation ocorre quando uma thread nunca obtém acesso a um recurso porque outras threads monopolizam o recurso.

Locais Potenciais no Código:

- Threads de clientes com menor prioridade ou que aguardam por locks muito tempo.

Mitigação:

- Usar mutexes justos (fair locks) se disponíveis.
- Evitar seções críticas longas.
- Garantir que todas as threads tenham oportunidade de execução.

---

3. Estratégias de Mitigação Adotadas

- **Mutexes:**
  O uso de mutexes em libtslog.cpp para proteger operações de logging.
- **Estruturas Thread-Safe:**
  Garantir que listas ou buffers compartilhados entre threads sejam protegidos.
- **Design Simples:**
  Evitar dependências circulares entre locks.

---

4. Sugestões de Melhoria

- Revisar o uso de variáveis globais e garantir proteção adequada.
- Adotar ferramentas de análise estática (ex: ThreadSanitizer) para detectar condições de corrida.
- Documentar claramente as regiões críticas no código.

---

5. Referências de Código

- libtslog.cpp: Funções de logging e uso de mutexes.
- conn.cpp: Gerenciamento de conexões e possíveis regiões críticas.
- server.cpp: Criação e gerenciamento de threads para clientes.

6. Conclusão

O sistema implementa mecanismos básicos de proteção contra problemas de concorrência, mas recomenda-se revisão contínua e uso de ferramentas automáticas para garantir robustez. O uso de mutexes e práticas de programação segura são essenciais para evitar race conditions, deadlocks e starvation.