13 - Concurrency

2025-10-03 17:45 pm

- Escrever código com *Concurrency* limpo é muito difícil
- Concurrency desacopla **o quê do quando
  - Em single-thread, estão acoplados e é fácil de saber o estado do sistema
  - Desacoplamento melhora estrutura e velocidade do código.
- **Mitos:**
  - ❌ Concurrency sempre melhora o desempenho
    - Pode ou não melhorar
  - ❌ Design não muda quando escrevemos código com concurrency
    - Muda muito
  - ❌ Entender concurrency não é importante quando se lida com um container como Web Frameworks
    - É bom entender como funcionam para evitar problemas
- **Questões:**
  - Concurrency gera overhead de tempo de código
  - Concurrency correta é complexo
  - Bugs de Concurrency nem sempre são reproduzíveis
  - Concurrency requer mudanças fundamentais no design

Desafios:

- Há mais de um caminho possível para o mesmo código
  - Resultados diferentes possíveis

Princípios

- Há alguns princípios para proteger código desses problemas
- **SRP**:
  - Código de concorrência pode se misturar mas requer tratamento específico
  - Recomendação é separar concurrency do resto do código
  - **Limitar escopo dos dados:**
    - *synchronized* keyword permite um único thread por vez para proteger alguma seção crítica de dado compartilhado por threads (em Java)
      - Equivalente em Python seria usando um *Lock* com um *with*
  - **Usar cópias dos dados**:
    - Evitar compartilhamento de dados entre threads
    - Tratar variáveis como read-only e gerar cópias
  - **Threads devem ser o mais independentes possível:**
    - Thread deve existir por si só sem depender de nada externo a ela
  - *Ideal é tentar dividir os dados em subconjuntos independentes*

Recursos

- Recomendações:

1. **Use thread-safe collections**
    - e.g. em Java usar `ConcurrentHashMap` em vez de `Hashmap`
2. **Use o framework executor para realizar tarefas não relacionadas**
3. **Use soluções não-bloqueantes quando possível**
4. **Várias classes não são thread-safe**

- Modelos de execução:
    - **Producer-Consumer**:
        - Thread producer insere tarefas numa queue e um consumer thread adquire aquela tarefas da fila e a executa.
        - Queue é um recurso limitado, tem um tamanho máximo.
        - Requer coordenação entre consumer e producer para agir no tempo certo.
    - **Reader-Writer:**
        - Reader não pode ler algo que o writer está escrevendo e vice-versa.
        - Writer bloqueia os readers.
        - Precisa equilibrar os dois para evitar starvation de um ou do outro.
    - **Dining Philosophers:**
        - Competição por recursos.
        - Pode gerar deadlock (um thread espera o outro acabar) ou livelock (thread são interrompidos continuamente por outros threads)
    - *Importante aprender estes padrões e como solucioná-los*

- Se há dois métodos synchronized na mesma classe, pode haver um bug.
    - Não usar mais de um método em um objeto compartilhado.
        - Se for necessário, requer soluções com locking
    - Synchronized deve ser evitado quando possível pois gera gargalos.
        - A seções synchronized devem ser as menores possíveis

- Shutdown correto é difícil
    - Fácil de gerar deadlock
    - Ideal é pensar nisso desde o começo.

Testing

- Testar é difícil
- Ideal é ter muitos testes e rodar com frequência
- Recomendações
    - **Não ignorar testes que falham uma vez e passam depois.**
        - Pode indicar um problema maior de threading.
    - **Começar fazendo antes funcionar a parte do código single-threaded.**
    - **Fazer código multithread ser pluggable e tunable**
        - Permite rodar de várias formas diferentes para testar.
    - **Tentar forçar erros de concurrency**
        - Rodar com mais threads que processadores
        - Rodar em plataformas diferentes com frequência.
        - Erros podem acontecer só em algumas situações bem específicas e de acaso.
            - Wait, yield e sleep e afins podem variar ordem para forçar isso
            - **Manual:**
                - Colocar esses comandos manualmente.
                - Difícil e demorado.
            - **Automatizado:**

- Frameworks para instrumentação disso.
- Jiggling automático dos threads para alterar ordem.