

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ - PUCPR

DIEGO NUNES

GUSTAVO H. SCHOTT

LEONARDO PEREIRA

MECANISMOS DE CONCORRÊNCIA, IMPASSE, STARVATIONS, RACE E DEADLOCK

CURITIBA

2025

1. INTRODUÇÃO

Este relatório apresenta uma análise detalhada dos principais problemas de concorrência, utilizando três abordagens complementares: o modelo teórico do Jantar dos Filósofos, uma demonstração prática de condição de corrida (race condition) em um contador compartilhado, e a reprodução e correção de um deadlock entre dois locks no Java. Cada uma dessas partes ilustra um tipo distinto de falha provocada pela execução paralela sem coordenação adequada. O objetivo deste documento é demonstrar total domínio dos conceitos de impasse, fome, race condition, sincronização, visibilidade de memória, happens-before e hierarquia de recursos.

2. DESENVOLVIMENTO

2.1 O PROBLEMA DOS FILÓSOFOS

O Jantar dos Filósofos é um dos modelos clássicos para representar problemas de sincronização. Ele mostra como processos concorrentes podem competir por recursos finitos, causando inconsistências, travamentos e falta de progresso. Cada filósofo representa uma thread; os garfos representam recursos que só podem ser usados por um processo de cada vez.

Quando todos os filósofos tentam comer ao mesmo tempo e cada um pega primeiro o garfo à sua esquerda, surge a situação de impasse. Cada filósofo segura um garfo e espera pelo outro que nunca será liberado, formando um ciclo de dependência. Esse ciclo caracteriza o deadlock. A análise do problema pode ser explicada pelas quatro condições de Coffman: exclusão mútua, manter-e-esperar, não preempção e espera circular. As três primeiras são inerentes ao problema; a quarta é a condição que pode ser quebrada para impedir o impasse.

Além do deadlock, o problema também apresenta a possibilidade de fome (starvation), quando um filósofo nunca consegue acesso aos dois garfos, mesmo que o sistema continue funcionando. Isso ocorre por falta de fairness na ordem de atendimento.

A solução adotada consiste em impor uma ordem global para a aquisição dos recursos. Cada filósofo identifica seus dois garfos e sempre tenta adquirir primeiro o garfo de menor índice e depois o de maior índice. Essa simples mudança elimina a condição de espera circular e, portanto, torna o deadlock impossível. Embora não elimine totalmente a fome, reduz drasticamente sua ocorrência e estabiliza o sistema.

2.2 Condição de Corrida, Happens-Before, Visibilidade e Uso de Semáforo

A segunda parte demonstra uma condição de corrida real usando o código `SemSemaforo.java`. O problema ocorre quando múltiplas threads incrementam simultaneamente uma variável global. A expressão `contador++` não é atômica; ela envolve leitura, soma e escrita. Quando essas operações são intercaladas entre várias threads, parte das atualizações é perdida. Como resultado, o valor final obtido é muito menor do que o esperado.

Essa inconsistência está diretamente relacionada à ausência de uma relação happens-before entre as threads. Sem mecanismos de sincronização, não há garantia de que as escritas feitas por uma thread serão visíveis às outras. A JVM e o processador podem reordenizar instruções, o que agrava a perda de previsibilidade. Assim, a race condition não é apenas uma disputa de tempo, mas também um problema de memória e ordenação.

A correção é implementada no código `Main.java` usando um semáforo binário com fairness ativado. O semáforo garante exclusão mútua, permitindo que apenas uma thread execute o incremento por vez. O parâmetro `true` no construtor ativa a política FIFO, prevenindo starvation entre as threads. Além disso, cada `release` estabelece um happens-before para o próximo `acquire`, garantindo visibilidade adequada das escritas. O resultado final passa a ser determinístico e igual ao esperado. Esse exemplo demonstra claramente como mecanismos de sincronização corrigem falhas de atomicidade e visibilidade em ambientes paralelos.

2.3 Deadlock Real entre Dois Locks e Prevenção por Hierarquia de Recursos

A terceira parte do trabalho apresenta um deadlock real entre dois locks, replicado no código `DeadlockDemo.java`. O problema ocorre quando duas threads tentam adquirir dois locks em ordens diferentes: uma thread segura `lockA` e aguarda `lockB`, enquanto a outra segura `lockB` e aguarda `lockA`. Nenhuma das duas pode prosseguir, criando um impasse definitivo.

Esse cenário é uma reprodução exata das quatro condições de Coffman. Há exclusão mútua devido ao uso de `synchronized`; manter-e-esperar, pois cada thread retém um lock enquanto solicita outro; não preempção, já que locks não podem ser retirados; e espera circular, que fecha o ciclo fatal. Dado que todas as condições estão presentes, o deadlock é inevitável.

A solução aplicada segue o mesmo princípio utilizado no problema dos filósofos: estabelecer uma hierarquia global para aquisição dos recursos. Os locks são ordenados pela identidade do objeto, e todas as threads passam a adquirir sempre primeiro o lock com menor identidade e, em seguida, o maior. Essa abordagem elimina a espera circular e, assim, torna o deadlock impossível. A hierarquia de recursos é uma técnica amplamente utilizada em sistemas operacionais e bancos de dados por ser simples, eficiente e formalmente segura.

3. Relação entre as três partes do trabalho

As três partes do trabalho estão relacionadas porque tratam de diferentes consequências da falta de coordenação entre threads. No problema dos filósofos, a ausência de uma política organizada leva ao impasse. No contador compartilhado, a ausência de sincronização gera inconsistência de dados. No deadlock entre dois locks, a falta de hierarquia provoca travamento do sistema.

O ponto comum é que todos esses problemas são resolvidos quando impomos ordem, visibilidade e previsibilidade à execução concorrente. Semáforos, fairness e hierarquia de recursos são ferramentas diferentes que atacam problemas

relacionados, oferecendo garantias específicas: exclusão mútua, eliminação de race conditions, prevenção de deadlocks e estabilidade na progressão das threads. Assim, o conjunto dos três exemplos permite entender como sistemas concorrentes devem ser projetados para evitar falhas clássicas de sincronização.

4. Conclusão

Conforme demonstrado nos vídeos, o estudo do conjunto das três partes (Jantar dos Filósofos, a race condition, o deadlock) demonstra de forma clara como é complexa a execução concorrente e os riscos associados à falta de coordenação. A eliminação de deadlocks, quando se evita a espera circular e por exclusão mútua, além do fairness e uso de mecanismos de sincronização. O objetivo esperado, foi demonstrar conhecimento sólido, além de, soluções práticas e aplicadas com técnicas formais e que são amplamente utilizadas em sistemas reais, algo que reforça a importância de protocolos bem estruturados, que garantam a eficiência e correção desses problemas em ambientes que dependem de multi-thread.

5. REFERÊNCIAS BIBLIOGRÁFICAS

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. Operating System Concepts. Disponível em: <https://www.os-book.com>. Acesso em: 14 nov. 2025.

ORACLE. Java Platform, Standard Edition – Concurrency and Multithreading Documentation. Disponível em: <https://docs.oracle.com/javase/specs/>. Acesso em: 14 nov. 2025.

DIJKSTRA, Edsger W. The Dining Philosophers Problem. Disponível em: <https://www.cs.utexas.edu/users/EWD/>. Acesso em: 14 nov. 2025.