

# TDE3

# Sistemas





# Introdução

Este trabalho aborda o problema clássico do *Jantar dos Filósofos*, ilustrando a ocorrência de deadlocks em sistemas concorrentes. Vamos explorar também condições de corrida em threads e apresentar soluções práticas usando hierarquia de recursos e semáforos para evitar impasses e garantir sincronização adequada.

# Jantar dos Filósofos



```
Constantes: N = 5
Recursos: Garfos[0..4]

Para cada Filósofo(i):
    // Define a ordem de pegar os garfos (menor primeiro)
    primeiro_garfo = min(i, (i + 1) % N)
    segundo_garfo = max(i, (i + 1) % N)

    Loop Infinito:
        pensar()
        estado[i] = "COM FOME"

        adquirir(Garfos[primeiro_garfo]) // Pega o menor índice
        adquirir(Garfos[segundo_garfo]) // Pega o maior índice

        estado[i] = "COMENDO"
        comer()

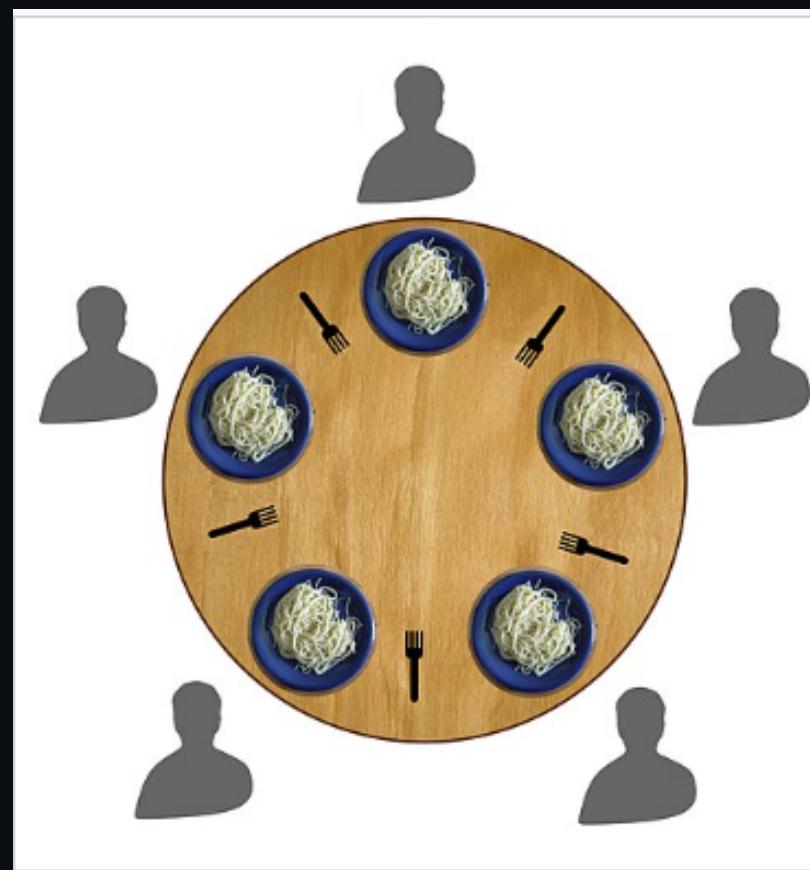
        liberar(Garfos[segundo_garfo])
        liberar(Garfos[primeiro_garfo])

        estado[i] = "PENSANDO"
```

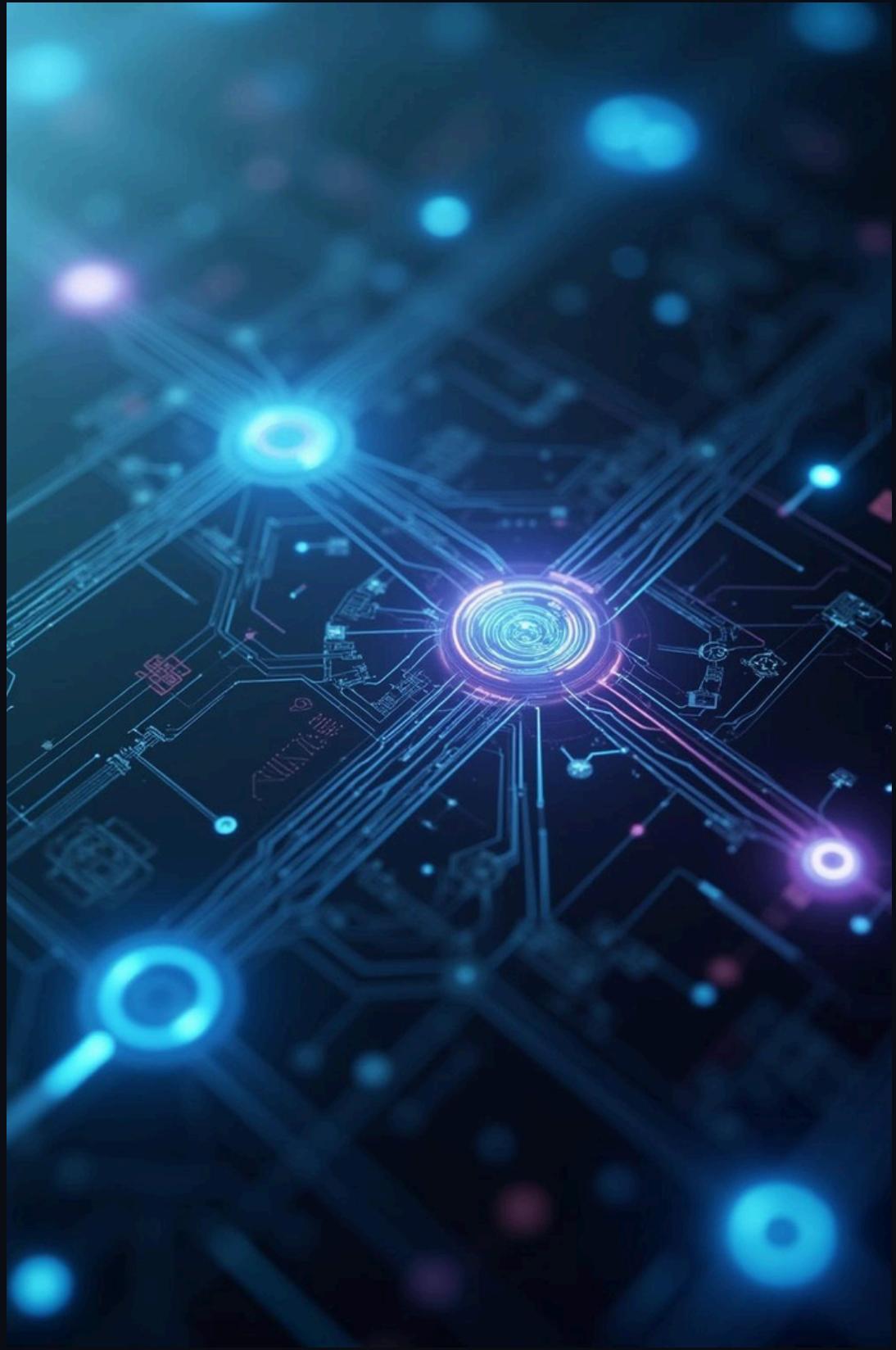
O que é Deadlock?

*Definição Simples:* Deadlock (ou Impasse) é uma situação em que dois ou mais processos (ou threads) ficam bloqueados indefinidamente, pois cada um está segurando um recurso e esperando por outro recurso que está sendo segurado por outro processo do grupo. Ninguém progride porque ninguém solta o que tem.

# Solução por Hierarquia de Recursos

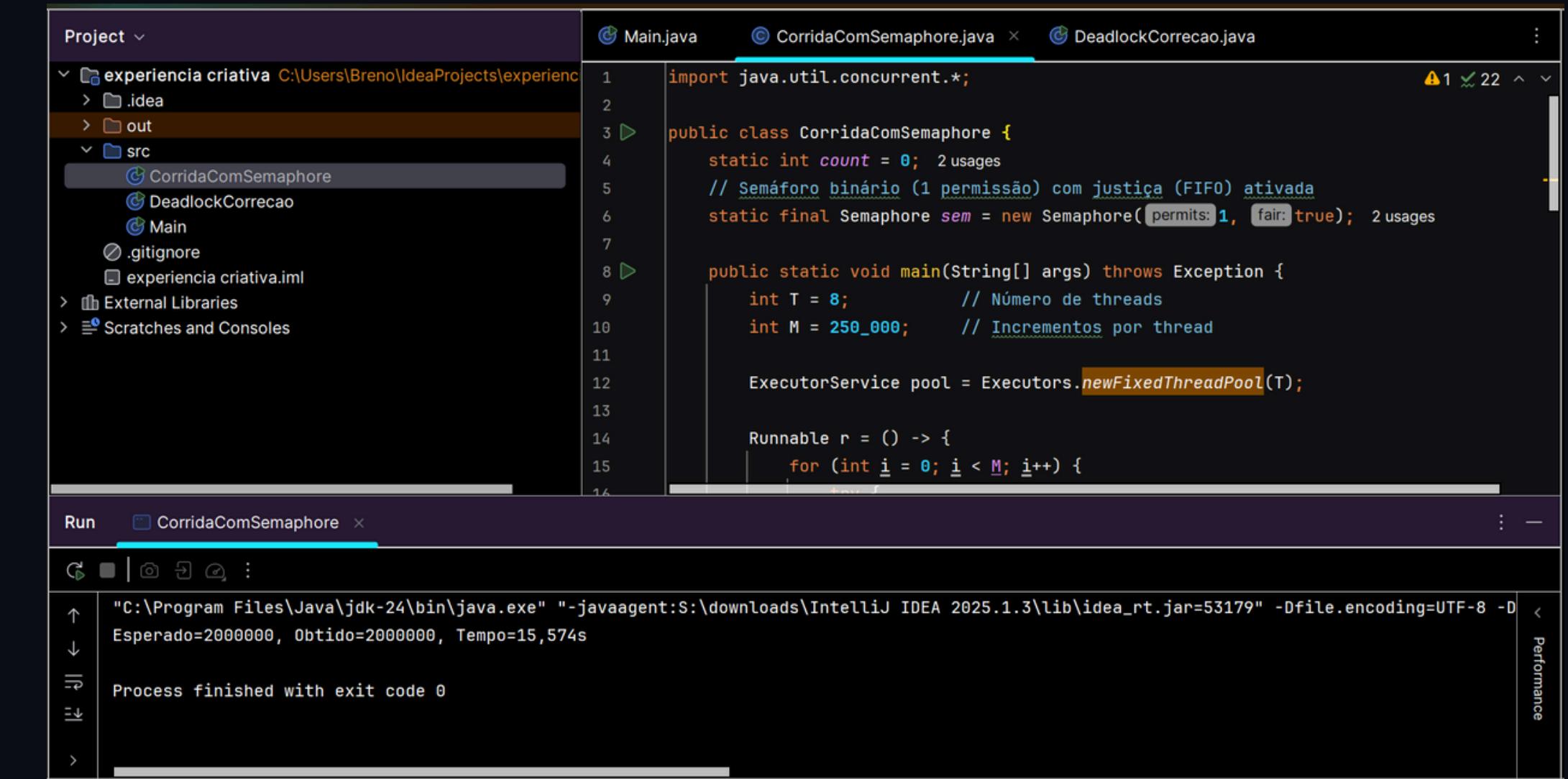


Para resolver isso, é necessário utilizar estratégia de Hierarquia de Recursos. Nós numeramos os garfos de 0 a 4. A lógica é a seguinte: todo filósofo deve pegar obrigatoriamente o garfo de menor número primeiro e só depois o de maior número. Isso quebra a condição de espera circular de Coffman, pois o último filósofo não conseguirá fechar o ciclo de dependência."



# Threads e Deadlock

# Uso de Semáforos para Sincronização



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "experiencia criativa". The "src" folder contains three files: CorridaComSemaphore.java, DeadlockCorrecao.java, and Main.java. The "CorridaComSemaphore.java" file is currently selected.
- Code Editor:** Displays the Java code for "CorridaComSemaphore". It imports java.util.concurrent.\* and defines a static final Semaphore "sem" with permits: 1 and fair: true. The main method creates an ExecutorService pool of 8 threads and a loop that increments a counter "count" 250,000 times per thread.
- Run Tab:** Set to run "CorridaComSemaphore".
- Output Tab:** Shows the command used to run the application: "C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:S:\downloads\IntelliJ IDEA 2025.1.3\lib\idea\_rt.jar=53179" -Dfile.encoding=UTF-8 -D Esperado=2000000, Obtido=2000000, Tempo=15,574s. It also indicates "Process finished with exit code 0".

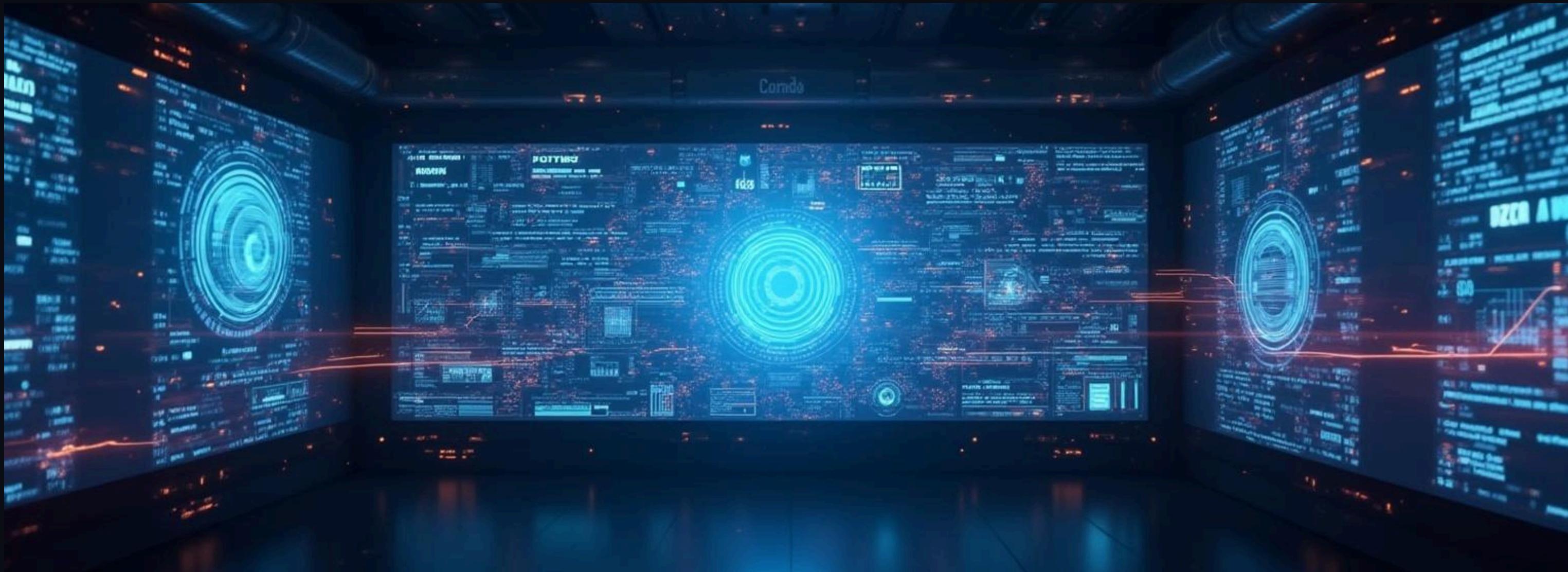
Semáforos são mecanismos que garantem **exclusão mútua**, permitindo que apenas uma thread acesse um recurso crítico por vez. O uso do modo *fair* cria uma fila FIFO, prevenindo a inanição das threads. Essa sincronização evita condições de corrida, embora possa impactar o desempenho em termos de throughput.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "experiencia criativa". The "src" folder contains three packages: "CorridaComSemaphore", "DeadlockCorrecao", and "Main". The "DeadlockCorrecao" package is currently selected.
- Main.java:** Contains the main class definition and its main method.
- CorridaComSemaphore.java:** Contains a static final Object named LOCK\_A.
- DeadlockCorrecao.java:** Contains the main logic for thread T1. It prints "T1 pegou A", sleeps for 50ms, and then prints "T1 pegou B -> T1 concluiu".
- Run Tab:** Shows the command used to run the application: "C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:S:\downloads\IntelliJ IDEA 2025.1.3\lib\idea\_rt.jar=53222" -Dfile.encoding=UTF-8 DeadlockCorrecao.
- Terminal Output:** Displays the execution results:
  - Up arrow: T1 pegou A
  - Down arrow: T1 pegou B -> T1 concluiu
  - Left arrow: T2 pegou A
  - Right arrow: T2 pegou B -> T2 concluiu

# Análise e Correção do Deadlock

O deadlock ocorre quando coexistem quatro condições: exclusão mútua, hold and wait, não preempção e espera circular. A correção requer impor uma ordem fixa na aquisição dos recursos, eliminando a espera circular e prevenindo que as threads travem indefinidamente, garantindo a continuidade e a estabilidade do sistema.



# Conclusões

A aplicação da hierarquia de recursos e o uso de semáforos são fundamentais para evitar deadlocks e condições de corrida em sistemas operacionais. Compreender as condições de Coffman permite diagnosticar e corrigir esses impasses, assegurando a eficiência e confiabilidade dos processos concorrentes. A prática e o controle rigoroso são essenciais para o sucesso em ambientes multithread.

• •

---

# Obrigado!

CREDITS: Breno Augusto Rocha