## Implementação corrigida

A correção consistiu em impor uma ordem fixa de aquisição dos locks, de forma que todas as threads adquiram os recursos sempre na mesma sequência (primeiro `LOCK_A`, depois `LOCK_B`). Essa hierarquia de recursos impede que duas threads adquiram os locks em ordem oposta, eliminando a condição de espera circular, uma das quatro condições necessárias para deadlock segundo Coffman.

**As quatro condições de Coffman:**

- **Exclusão mútua:** um recurso só pode ser usado por uma thread por vez.
- **Manter-e-esperar (hold and wait):** uma thread mantém recursos já adquiridos enquanto espera por outros.
- **Não preempção:** recursos não podem ser retirados de uma thread até que ela os libere voluntariamente.
- **Espera circular:** existe um ciclo de threads em que cada uma espera por um recurso mantido pela próxima, formando um loop fechado de dependência.

Se todas essas condições ocorrerem simultaneamente, o deadlock é possível; eliminar qualquer uma delas impede o impasse.

No caso onde foi aplicada a solução, sem a possibilidade de formar um ciclo de dependências, o sistema não entra em impasse e todas as threads conseguem progredir.
Foi reduzida a ocorrência do *manter-e-esperar*, pois nenhuma thread pode ficar segurando um lock "superior" enquanto aguarda um lock "inferior". A ordem fixa garante que não há deadlock.

---

## Relação com o problema dos filósofos

No problema do jantar dos filósofos, o deadlock surge quando cada filósofo pega primeiro o garfo da esquerda e fica esperando o da direita, criando um ciclo de espera. Uma solução comum é impor uma hierarquia: cada filósofo deve pegar os garfos sempre na ordem do menor para o maior.

No meu código, os garfos correspondem aos locks, e os filósofos às threads.
Ao impor uma ordem global de aquisição dos recursos, o ciclo de dependência é removido, o que elimina a espera circular e previne o deadlock.