# Sistemas Operacionais I

# Aula passada

- Deadlock;
- Modelo de Sistema;
- Caracterização do Deadlock;
- Grafo de Alocação de Recursos.

# Agenda

- Métodos para a Manipulação de Deadlocks;
- Prevenção/Impedimento de Deadlocks;
- Detecção e Recuperação de Deadlocks.

# Métodos para a Manipulação de Deadlocks

# Métodos para a Manipulação de Deadlocks

Podemos lidar com o problema do **deadlock** de uma entre três maneiras:

1. Podemos usar um **protocolo** para impedir ou evitar a ocorrência de **deadlock** , assegurando que o sistema nunca entrará em estado de **deadlock**;
2. Podemos permitir que o sistema entre em estado de **deadlock**, **detecte-o** e se **recupere**;
3. Podemos **ignorar** o problema completamente e fingir que **deadlock** nunca ocorrem no sistema.

# Métodos para a Manipulação de Deadlocks

3. Podemos **ignorar** o problema completamente e fingir que **deadlock** nunca ocorrem no sistema:
    a. Embora o risco de **deadlock** seja uma questão importante, os sistemas operacionais de mercado (Windows, Linux, MacOS, etc;) adotam a solução mais simples;
    b. Devido ao **custo computacional** necessário ao tratamento de impasses e à sua forte dependência da lógica das aplicações envolvidas, os projetistas de sistemas operacionais normalmente preferem deixar a gestão por conta dos desenvolvedores de aplicações.

# Prevenção de Deadlocks

# Prevenção de Deadlocks

- As técnicas de prevenção buscam garantir que **deadlock** nunca possam ocorrer no sistema;
- Para alcançar esse objetivo, a **estrutura do sistema** e a **lógica das aplicações** devem ser construídas de forma que as **quatro condições** fundamentais para a ocorrência de **deadlock.**

1. **Exclusão mútua:** o acesso aos **recursos** deve ser feito de forma **mutuamente exclusiva**, controlada por **semáforos** ou **mecanismos equivalentes**.
2. **Posse e espera:** uma **tarefa** pode **solicitar** o acesso a **outros recursos** sem ter de **liberar** os recursos que já detém;
3. **Não-preempção:** uma tarefa somente libera os recursos que detém quando assim o decidir, e não os perde de forma imprevista (ou seja, o sistema operacional não retira à força os recursos alocados às tarefas);
4. **Espera circular:** existe um ciclo de esperas pela liberação de recursos entre as tarefas envolvidas: **t1** → t2 → t3 → · · · → tn → **t1**.

# Prevenção de Deadlocks

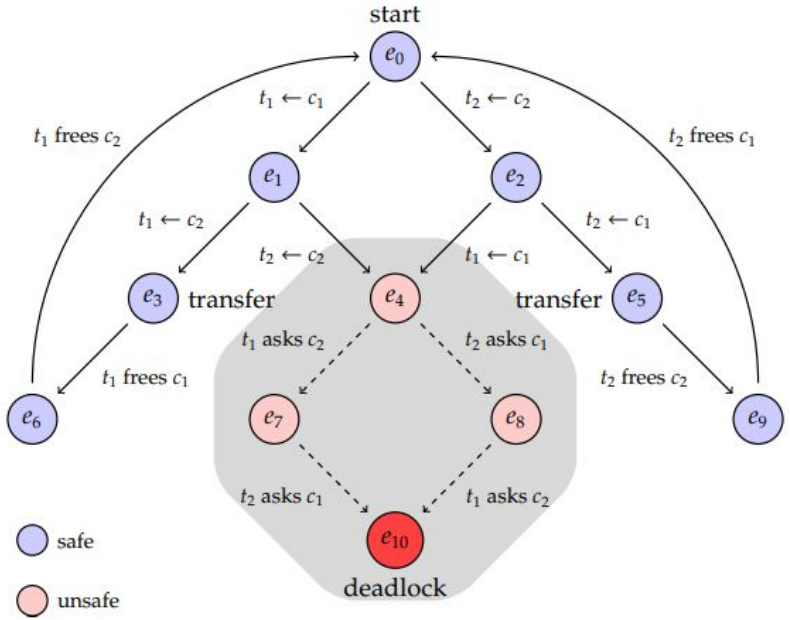
1. **Exclusão mútua:**
    a. A condição de exclusão mútua deve estar presente. Isto é, pelo menos um recurso deve ser não compartilhável;
    **b. Em geral, porém, não podemos prevenir a ocorrência de deadlocks negando a condição de exclusão mútua, porque alguns recursos são intrinsecamente não compartilháveis;**

# Prevenção de Deadlocks

2. **Posse e espera:**
    a. Outra forma de tratar os **deadlock** preventivamente consiste em **acompanhar** a **alocação** dos recursos às tarefas e, de acordo com algum **algoritmo**, **negar acessos** de recursos que possam levar a **deadlock**;
    b. Uma noção essencial nas técnicas de impedimento de impasses é o conceito de **estado seguro**;
        i. O conjunto de todos os estados possíveis do sistema durante sua execução forma um grafo de estados, no qual as arestas indicam as alocações e liberações de recursos;
        ii. Um determinado **estado** é considerado **seguro** se, a partir dele, é possível concluir as tarefas pendentes. Caso o estado em questão somente leve a **deadlock**, ele é considerado um **estado inseguro**;
        iii. As técnicas de impedimento de **deadlock** devem portanto manter o sistema sempre em um **estado seguro**, evitando entrar em **estados inseguros.**

# Prevenção de Deadlocks

A técnica de impedimento de impasses mais conhecida é o **algoritmo do banqueiro**, criado por Dijkstra em 1965;



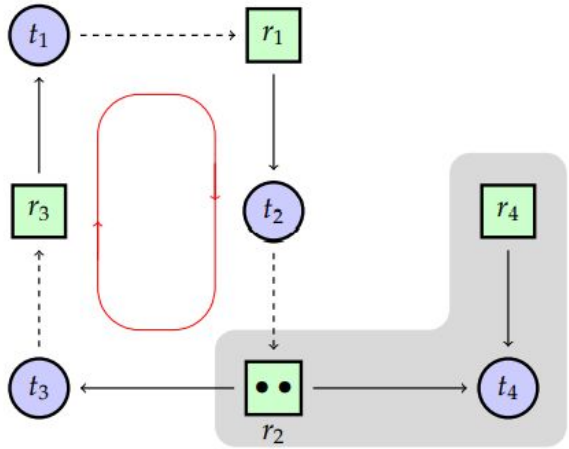

**Grafo de estados** do sistema de transferências com duas tarefas.

# Prevenção de Deadlocks

3. **Não-preempção:**
    a. A terceira condição necessária para a ocorrência de **deadlock** é que **não haja preempção** de recursos que já tenham sido alocados;
    b. Se um processo está retendo alguns recursos e solicita outro recurso que não possa ser alocado imediatamente a ele (isto é, o processo deve esperar), então todos os recursos que o processo esteja retendo no momento **sofrem preempção**.

# Prevenção de Deadlocks

4. **Espera circular:**
    a. Uma forma de assegurar que essa condição jamais ocorra é impor **uma ordem absoluta a todos os tipos de recursos** e requerer que cada processo solicite recursos em uma ordem de enumeração crescente;
    b. Essa solução também pode ser aplicada ao problema do jantar dos filósofos, fazendo com que os filósofos pegue os palitos em ordem crescente



# Detecção e Recuperação de Deadlocks

# Detecção e Recuperação de Deadlocks

Nesta abordagem, **nenhuma medida preventiva é adotada** para prevenir ou evitar **deadlock**. As tarefas executam normalmente suas ações, alocando e liberando recursos conforme suas necessidades.

Quando ocorrer um **deadlock**, o sistema deve detectá-lo, determinar quais as tarefas e recursos envolvidos e tomar medidas para desfazê-lo.

Para aplicar essa técnica, duas questões importantes devem ser respondidas:

1. Como detectar os impasses?
2. E como resolvê-los?

# Detecção e Recuperação de Deadlocks

A detecção de **deadlock** pode ser feita através da **inspeção do grafo de alocação de recursos**, que deve ser mantido pelo sistema e **atualizado** a cada **alocação** ou **liberação** de recurso;

Um **algoritmo de detecção de ciclos** no grafo deve ser executado periodicamente, para verificar a presença das dependências cíclicas que podem indicar **deadlock**.

Alguns problemas decorrentes dessa estratégia são o **custo de manutenção contínua** do grafo de alocação e, sobretudo, o custo de sua análise: algoritmos de busca de ciclos em grafos têm **custo computacional elevado**.

# Detecção e Recuperação de Deadlocks

Uma vez detectado um **deadlock** e identificadas as **tarefas** e **recursos** envolvidos, o sistema deve proceder à resolução, que pode ser feita de duas formas:

1. **Eliminar tarefas**: uma ou mais tarefas envolvidas no **deadlock** são eliminadas, liberando seus recursos para que as demais tarefas possam prosseguir. A escolha das tarefas a eliminar deve levar em conta vários fatores, como o tempo de vida de cada uma, a quantidade de recursos que cada tarefa detém, o prejuízo para os usuários que dependem dessas tarefas, etc.

2. **Retroceder tarefas**: uma ou mais tarefas envolvidas no **deadlock** têm sua execução parcialmente desfeita (uma técnica chamada **rollback**), de forma a fazer o sistema **retornar a um estado seguro** anterior ao **deadlock**.

    a. Para retroceder a execução de uma tarefa, é necessário salvar periodicamente seu estado, de forma a poder recuperar um estado anterior quando necessário.

    b. Operações envolvendo a rede ou interações com o usuário podem ser muito difíceis ou mesmo impossíveis de retroceder: como desfazer o envio de um pacote de rede, ou a reprodução de um arquivo de áudio na tela do usuário?