SISTEMAS OPERACIONAIS II

Prof. Renato Jensen

Impasses (Deadlocks)

- **Princípios Básicos**
  - **Recursos**
    - Um recurso pode ser um dispositivo de hardware (por exemplo, uma unidade de fita) ou um conjunto de informações (por exemplo, um registro em um banco de dados)
    - A sequência de eventos para utilizar um recurso é:

      solicitar → utilizar → liberar
  - **Impasse**
    - Um conjunto de processos está em um impasse se cada processo no conjunto está esperando um evento que somente outro processo no conjunto pode causar

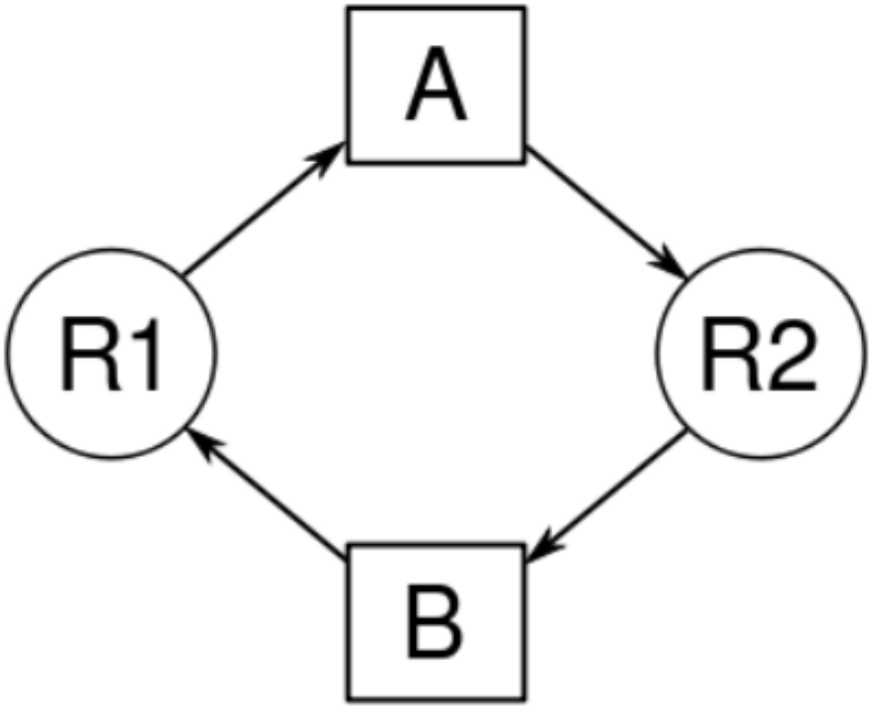
Impasses (Deadlocks)

- **Condições para um Impasse**

  Quatro condições devem ser verdadeiras para ocorrer um impasse:

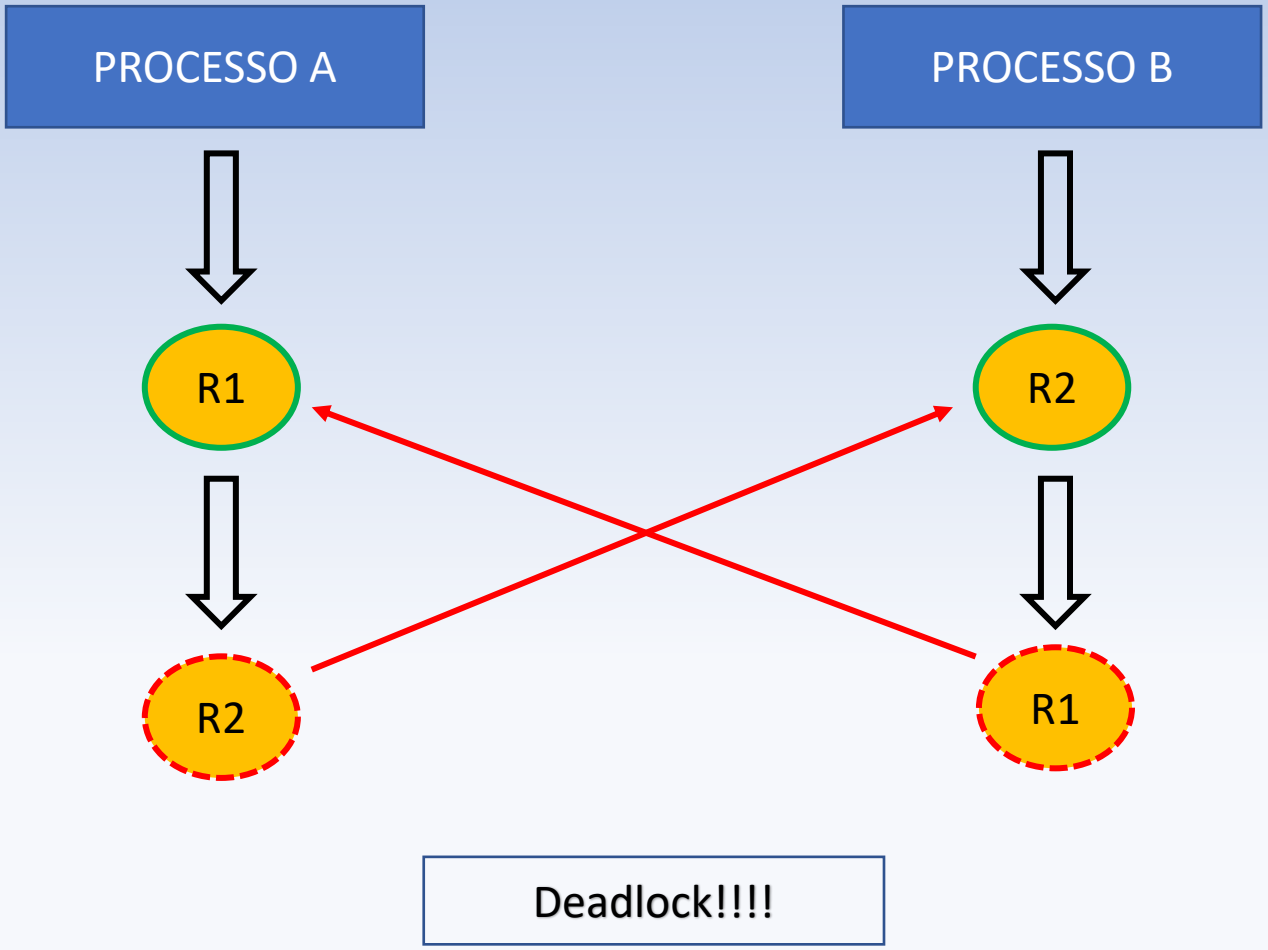
  - **Exclusão mútua**: o recurso está alocado a somente um processo ou está disponível
  - **Segura e espera**: processo que está segurando um recurso pode solicitar novos recursos
  - **Nenhuma preempção**: somente o processo que está segurando um recurso pode liberá-lo
  - **Espera circular**: em dois ou mais processos, cada um deles está esperando um recurso segurado pelo próximo membro da cadeia circular

Impasses (Deadlocks)



Exemplo de representação de *deadlock* em grafos de alocação de recursos, com dois processos **A** e **B**, e dois recursos **R1** e **R2**.

Impasses (Deadlocks)



Impasses (Deadlocks)

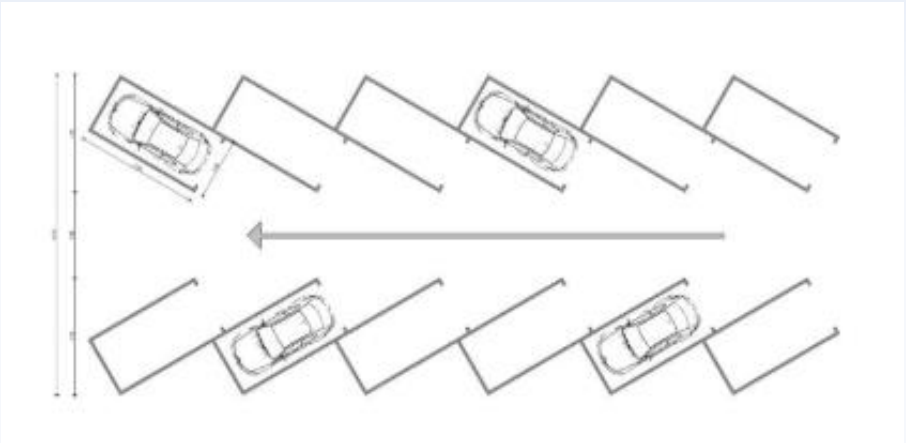
- **Estratégias para Lidar com Impasses**
  - **Ignorar o problema** (algoritmo do Avestruz). Ex.: UNIX.
  - **Detecção e recuperação**. Ex.: bancos de dados.

Impasses (Deadlocks)

- **Estratégias para Lidar com Impasses**
  - **Prevenção**. Não deixar ocorrer pelo menos uma das quatro condições de impasse:
    - Eliminar **exclusão mútua**: seria permitir que qualquer recurso pudesse ser usado por dois ou mais processos simultaneamente. Porém se deixarmos dois processos gravarem em uma mesma unidade de fita seria totalmente caótico (→ inviabiliza).
    - Eliminar **segura e espera**: uma solução para isso seria todos os processos solicitarem todos os recursos necessários antes de iniciar a execução; ou aloca tudo ou não aloca nada. Mas nem todos os processos sabem no seu início quantos recursos eles necessitarão e também os recursos seriam mal utilizados (→ inviabiliza).
    - Eliminar **nenhuma preempção**: processos poderiam liberar qualquer recurso alocado por outro processo. Porém tirar uma impressora de um processo no meio da sua impressão seria muito problemático (→ inviabiliza).
    - Eliminar **espera circular**: uma solução seria todos processos solicitarem os recursos sempre na mesma ordem; nesse caso o impasse nunca ocorreria. Se for possível de ser implantada, essa solução é a mais próxima de ser viável.

Impasses (Deadlocks)

- **Controle de Concorrência**
  - **Mutex ou Exclusão Mútua**: usado para garantir a exclusão mútua no acesso a recursos compartilhados. Permite somente um acesso por vez e se o recurso estiver em uso quem fez a solicitação aguarda em uma fila sua liberação.
  - **Semáforo**: permite controlar o número de acessos simultâneos a um recurso compartilhado. Seu valor inicial indica o número de acessos simultâneos possíveis. Por exemplo, um estacionamento com um número limitado de vagas. Um semáforo de valor inicial 1 é um um Mutex.



Impasses (Deadlocks)

- **Controle de Concorrência**

  - **Seção Crítica**: uma **seção crítica** - também conhecida por **região crítica** - é uma área de código de um algoritmo que utiliza dados ou recursos compartilhados, tais como variáveis, tabelas de banco de dados ou arquivos. Esse trecho de código não pode ser executado concorrentemente.

  - **Monitor**: um monitor é um conjunto de procedimentos, variáveis e estrutura de dados, todas agrupadas em um módulo especial. Os processos podem chamar os procedimentos do monitor sempre que desejarem, mas nunca poderão acessar diretamente seus recursos internos. Apenas um processo pode estar ativo dentro do monitor em um determinado instante.

Impasses (Deadlocks)

O Problema Clássico dos Filósofos Jantando

Cinco filósofos gastam a vida pensando e comendo. Os filósofos compartilham uma mesa redonda, cercada por cinco cadeiras, cada uma pertencendo a um filósofo. No centro da mesa existe uma tigela de arroz, e a mesa está disposta com apenas cinco hashis (pauzinhos). Quando um filósofo pensa ele não interage com os colegas. De vez em quando, um filósofo tem fome e tenta pegar os dois hashis próximos a ele. Um filósofo só pode pegar um hashi de cada vez. Obviamente, ele não pode pegar um hashi que já esteja na mão de um vizinho. Quando um filósofo com fome tem dois hashis ao mesmo tempo, ele come sem largar os hashis. Quando termina de comer, ele coloca os dois hashis na mesa e recomeça a pensar.

Se simplesmente esperarmos que um dos hashis especificado esteja disponível para pegá-lo, teremos um impasse se os cinco filósofos pegarem seus hashis esquerdos simultaneamente.

Após o hashi esquerdo ser pego podemos verificar se o hashi direito está disponível; se ele não estiver devolvemos o hashi esquerdo à mesa e aguardamos um tempo determinado para tentar de novo. Mas novamente não conseguiremos progresso de os cinco filósofos iniciarem esse procedimento simultâneamente (ninguém conseguirá comer).

Se utilizarmos uma matriz de ***semáforos***, um por filósofo, controlando os estados de "pensando", "com fome" ou "comendo" e um filósofo só pode passar para o estado "comendo" se nenhum vizinho estiver comendo, o problema estará resolvido. Se o filósofo que quer comer é o 2 então seus vizinhos são o 1 e 3 e assim sucessivamente.