Sistemas Operacionais

Problemas clássicos de comunicação entre processos.

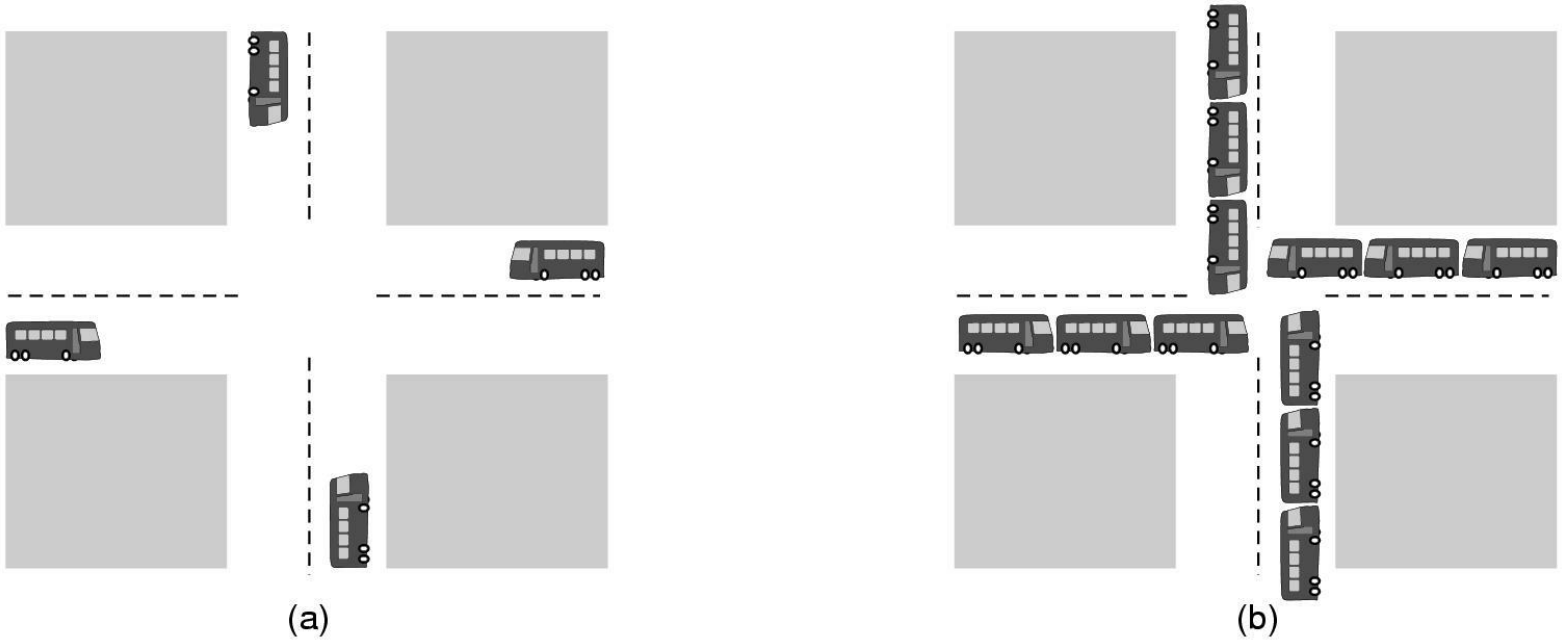
Prof. Felipe Leite Lobo
felipe.lobo@ufrr.br

Deadlocks

- Dispositivos e recursos são compartilhados a todo momento: impressora, disco, arquivos, etc...

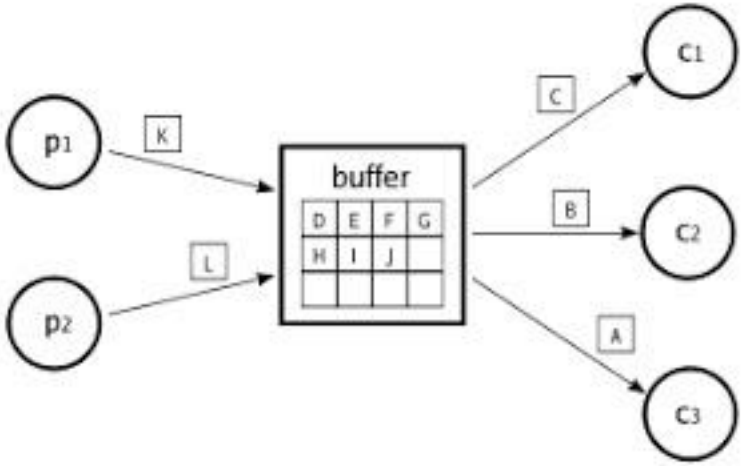
- ***Deadlock:*** processos ficam parados sem possibilidade de iniciar ou continuar seus processamentos.

Deadlocks



Fonte: Taenbaum

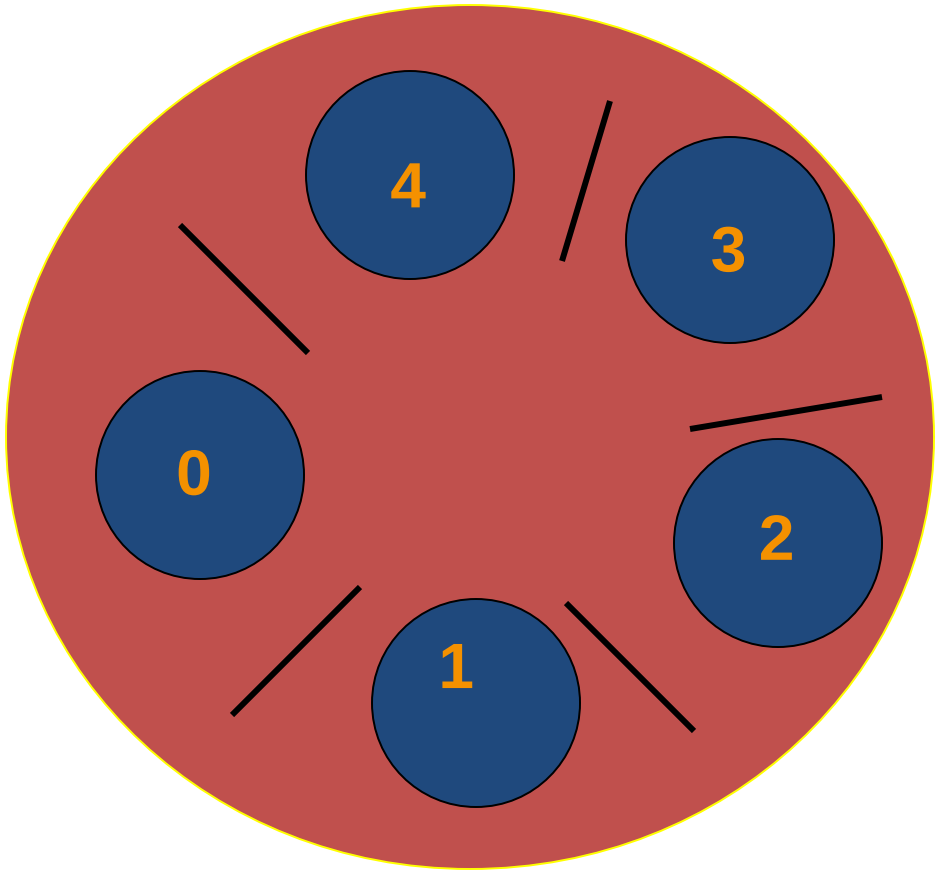
Produtor - Consumidor



Problemas clássicos de comunicação entre processos

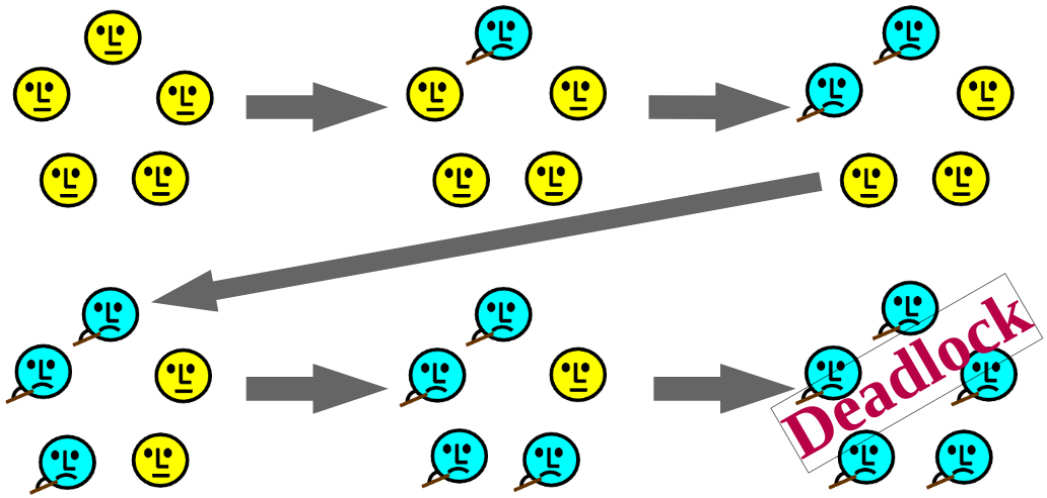
Problemas clássicos de comunicação entre processos

- Problema do Jantar dos Filósofos
  - Cinco filósofos desejam comer espaguete; No entanto, para poder comer, cada filósofo precisa utilizar dois garfos e não apenas um. Portanto, os filósofos precisam compartilhar o uso dos garfos de forma sincronizada.
  - Os filósofos comem e pensam.



Problemas clássicos de comunicação entre processos

- Problemas que devem ser evitados:
  - *Deadlock* – todos os filósofos pegam **um garfo** ao mesmo tempo.
  - *Starvation* – os filósofos **ficam indefinidamente pegando garfos simultaneamente.**



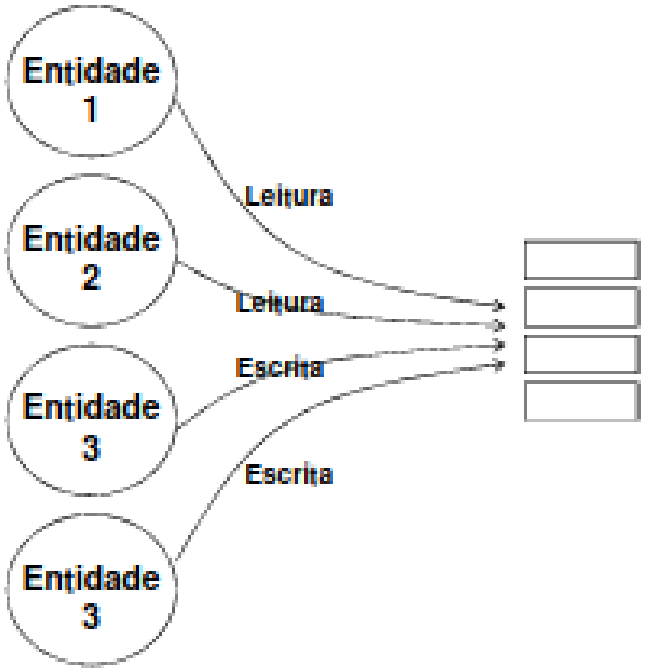
Problema dos Leitores / Escritores

- O problema dos Leitores e Escritores modela o acesso compartilhado a uma base de dados.

- Processos leitores e processos escritores competem por um acesso a essa base.

- É possível que vários processos leitores acessem a base ao mesmo tempo, no entanto, quando um processo escritor está escrevendo (modificando) a base de dados, nenhum outro processo pode realizar um acesso, nem mesmo um processo leitor.

Problema dos Leitores / Escritores



Problema do Barbeiro

- Na barbearia há um barbeiro, uma cadeira de barbeiro e ***n*** cadeiras para os clientes esperarem para ser atendidos;

- Quando não há clientes, o barbeiro senta-se na cadeira do barbeiro e dorme;

- Quando um cliente chega, ele precisa acordar o barbeiro para ser atendido. Se outros clientes chegarem enquanto o barbeiro estiver ocupado cortando o cabelo de algum cliente, eles se sentam se houver cadeiras disponíveis para clientes, senão eles vão embora se todas as cadeiras para clientes estiverem ocupadas.

Problema do Barbeiro

- Quando um cliente chega, ele precisa acordar o barbeiro para ser atendido. Se outros clientes chegarem enquanto o barbeiro estiver ocupado cortando o cabelo de algum cliente, eles se sentam se houver cadeiras disponíveis para clientes, senão eles vão embora se todas as cadeiras para clientes estiverem ocupadas.

Deadlocks

Deadlocks

Recursos: objetos acessados, os quais podem ser tanto de hardware quanto de software.

Preemptivos: podem ser retirados do processo sem prejuízos.

Ex: Memória;

CPU;

Não-preemptivos: não podem ser retirados do processo, pois causam prejuízos.

Ex: CD-ROM;

Unidades de fita;

Deadlocks ocorrem com recursos não-preemptivos.

Deadlocks

Operações sobre recursos/dispositivos:

- Requisição do recurso
- Utilização do recurso
- Liberação do recurso

Se o recurso requerido não está disponível, duas situações podem ocorrer:

- Processo que requisitou o recurso fica bloqueado até que o recurso seja liberado, ou;
- Processo que requisitou o recurso falha, e depois de um certo tempo tenta novamente requisitar o recurso.

Deadlocks

- Se vários processos tentam acessar os mesmos recursos, podem ocorrer situações onde a ordem de solicitação dos recursos pode conduzir ao um *deadlock*.

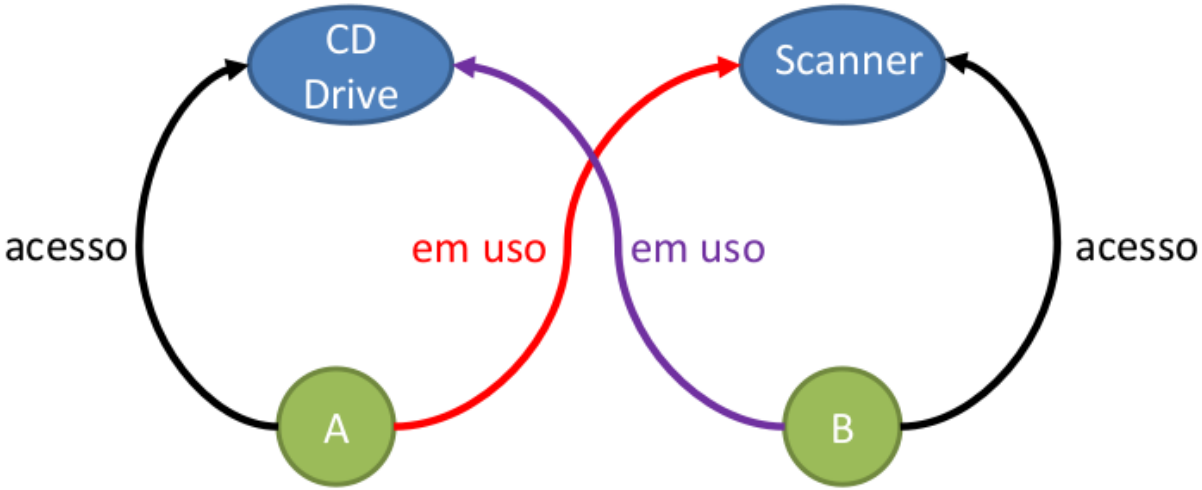
Definição formal:

> "Um conjunto de processos estará em situação de *deadlock* se todo processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer." (Tanenbaum,2010)

Deadlocks

- Quatro condições para que ocorra um *deadlock*:
  - **Exclusão mútua**: cada recurso pode estar somente em uma de duas situações: ou associado a um único processo ou disponível.
  - **Posse e espera (*hold and wait*):** processos que já possuem algum recurso podem requisitar outros recursos.
  - **Não-preempção:** recursos já alocados não podem ser retirados do processo que os alocou; somente o processo que alocou os recursos pode liberá-los.
  - **Espera Circular:** um processo pode esperar por recursos alocados ao processo seguinte da cadeia.

Deadlocks



- Como detectar Impasses?
- Como resolver Impasses?

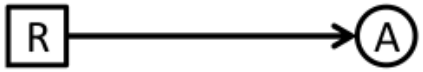
Deadlocks

- Quatro estratégias para tratar *deadlocks:*

1. Ignorar o problema;
2. Detectar e recuperar o problema;
3. Evitar dinamicamente o problema – alocação cuidadosa de recursos;
4. Prevenir o problema por meio da não satisfação de uma das três condições citadas anteriormente.

Modelagem de Impasses

- Útil modelar formalmente impasses;
- Principalmente para detecção de impasses;
- Utilização de Grafos Dirigidos;
- Dois tipos de estados;
- Dois tipos de arestas;



Recurso R alocado para o Proc. A

Proc. A requisita o recurso R

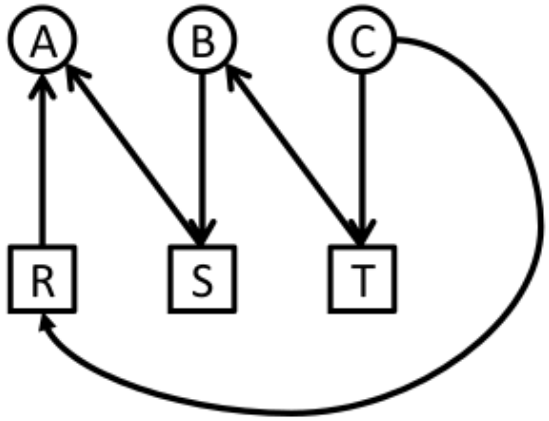
$$G = \{V, A\}$$

$$V = p_i \in P \vee r_i \in R$$

$$A = (p_i, p_j) / \; p_i \in P \wedge p_j \in R \vee p_i \in R \wedge p_j \in P$$

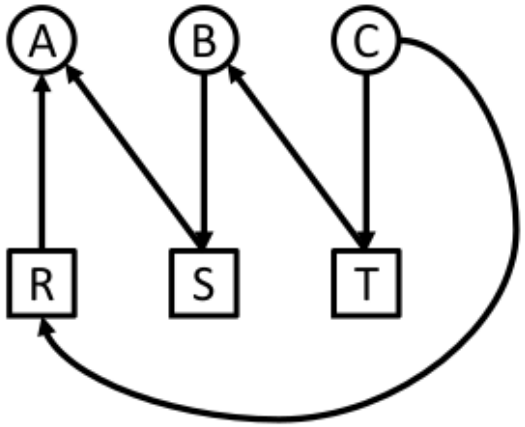
Modelagem de Impasses

- Processos não requisitam a alocação de recursos todos aos mesmo tempo;
- **Ordem de alocação** – a ordem em que os recursos são requisitados pelo processo;
- Ex: Alloc=({A,R},{A,S},{B,S},{B,T},{C,T},{C,R})



Modelagem de Impasses

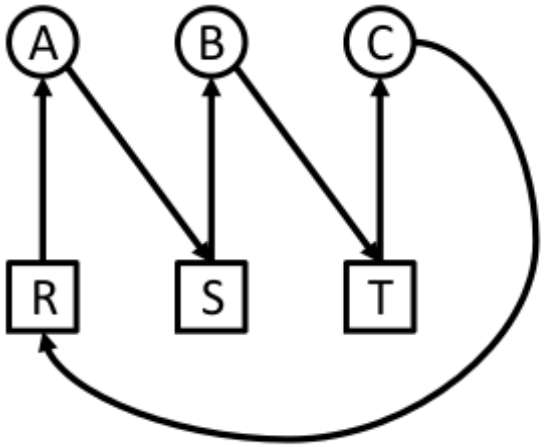
- Processos não requisitam a alocação de recursos todos aos mesmo tempo;
- **Ordem de alocação** – a ordem em que os recursos são requisitados pelo processo;
- Ex: Alloc=({A,R},{A,S},{B,S},{B,T},{C,T},{C,R})



Não há impasse
pois não há um ciclo!

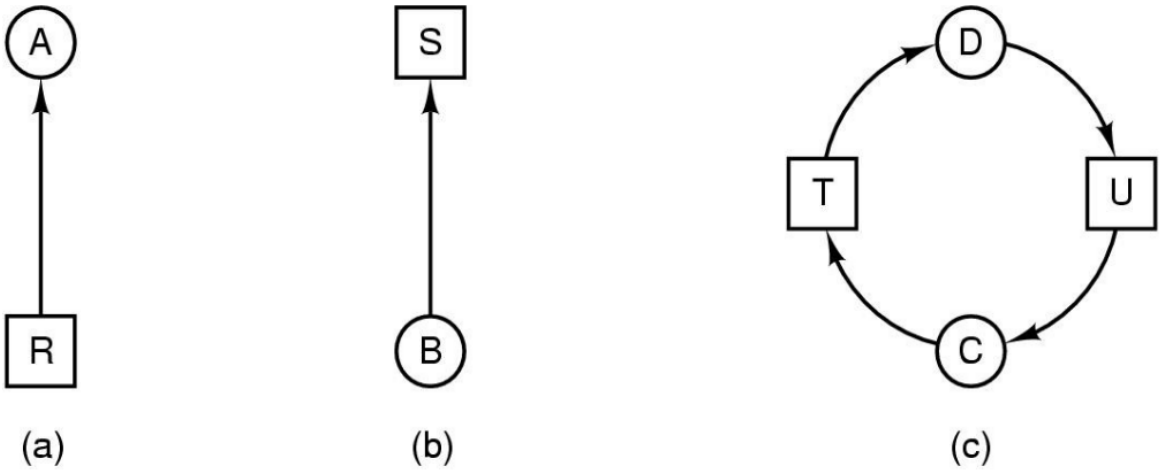
Modelagem de Impasses

- Ex: Alloc=({A,R},{B,S},{C,T},{A,S},{B,T},{C,R})



Há um empasse!
Moral da história: a ordem de alocação importa!

Deadlocks



a) **Recurso R** alocado ao **Processo A**
b) **Processo B** requisita **Recurso S**
c) ***Deadlock***

Deadlocks

1) **Ignorar o problema**:

- Frequência do problema;
- Alto custo – estabelecimento de condições para o uso de recursos;
- LINUX, UNIX e WINDOWS;
- Algoritmo do AVESTRUZ.

Deadlocks

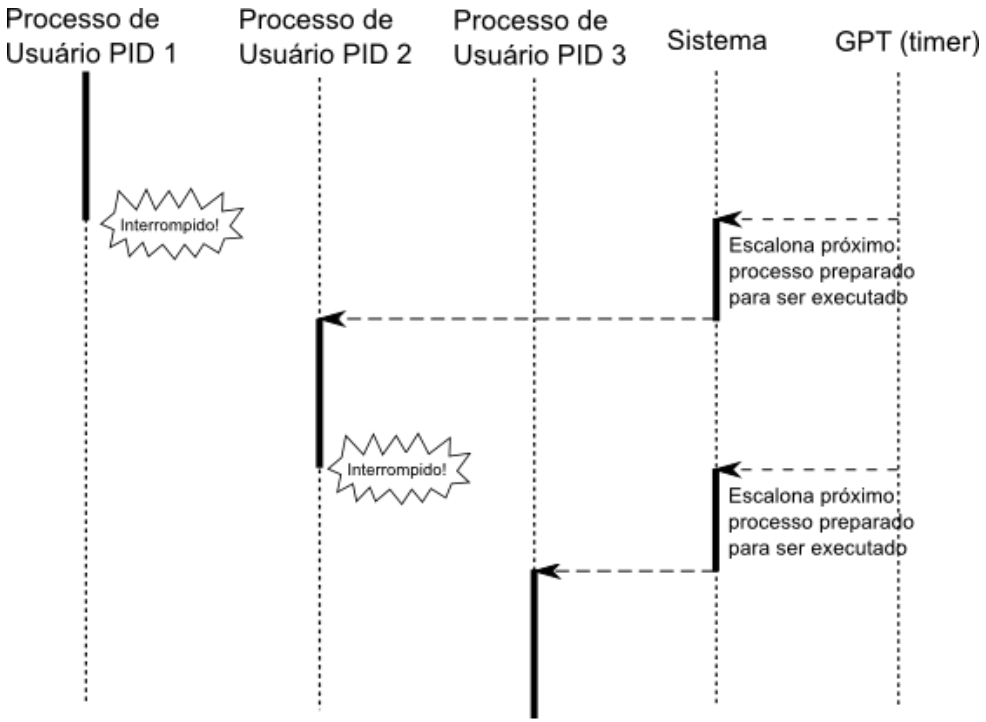
2) Detectar e Recuperar o problema:

- Processos estão com todos os recursos alocados.
- Procedimento: Permite que os *deadlocks* ocorram, tenta detectar as causas e solucionar a situação.
- Algoritmos:
  - Detecção com um recurso de cada tipo;
  - Detecção com vários recursos de cada tipo;

Deadlocks

2) Detectar e Recuperar o problema:

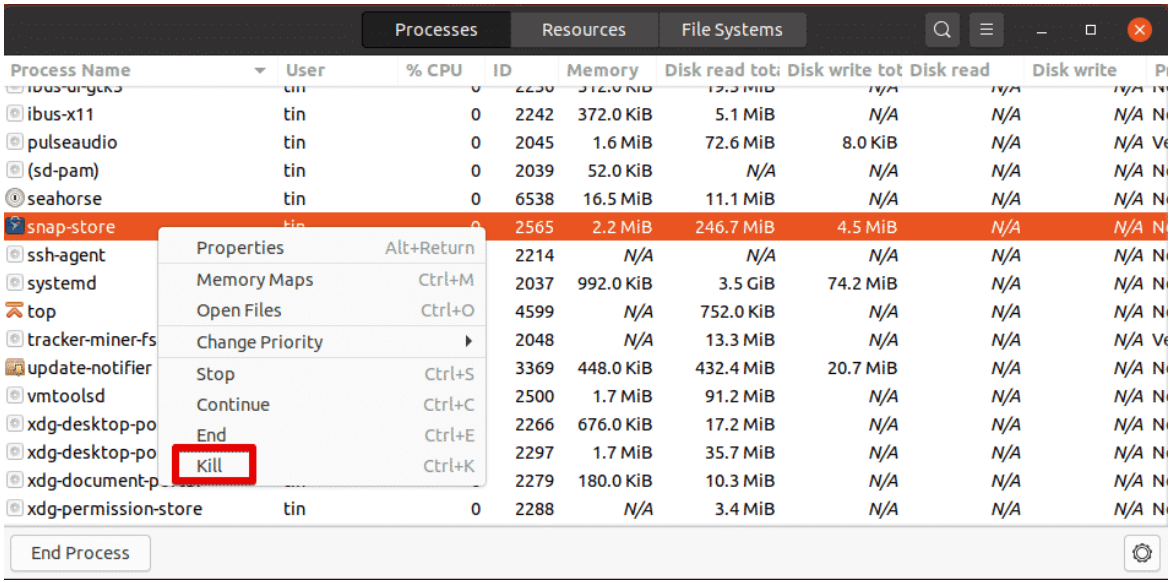
- Algoritmos:
  - Recuperação por meio de preempção;



Deadlocks

2) Detectar e Recuperar o problema:

- Algoritmos:
  - Recuperação por meio de *rollback* (volta ao passado);
  - Recuperação por meio de eliminação de processos.



Deadlocks

3) Evitar dinamicamente o problema:

- Alocação individual de recursos → à medida que o processo necessita.
- Soluções também utilizam matrizes.
- Escalonamento cuidadoso → alto custo.

  Conhecimento prévio dos recursos que serão utilizados.
- Algoritmos:

  Banqueiro para um único tipo de recurso.

  Banqueiro para vários tipos de recursos.

Deadlocks

Disponível. Um vetor de tamanho m indica o número de recursos disponíveis de cada tipo. Se ***Disponível***$[j]$ é igual a k, então k instâncias do tipo de recurso R_j estão disponíveis.

Max. Uma matriz $n \times m$ define a demanda máxima de cada processo. Se ***Max***$[i][j]$ é igual a k, então, o processo P_i pode solicitar, no máximo, k instâncias do tipo de recurso R_j.

Alocação. Uma matriz $n \times m$ define o número de recursos de cada tipo correntemente alocados a cada processo. Se ***Alocação***$[i][j]$ é igual a k, então o processo P_i tem correntemente alocadas k instâncias do tipo de recurso R_j.

Deadlocks

Temos 3 tipos de recursos com quantidades:

- R(1) = 9, R(2) = 3, R(3) = 6

E temos 4 processos com estado inicial:

	Claimed			Allocated			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	1	1	2
P2	6	1	3	5	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

- Suponha que P2 solicite Q = (1,0,1). A solicitação deve ser atendida?

Deadlocks

Possui

Máximo de linha de crédito = 22

A	0	6
B	0	5
C	0	4
D	0	7

Livre: 10

Seguro

A	1	6
B	1	5
C	2	4
D	4	7

Livre: 2

Seguro

A	1	6
B*	2	5
C	2	4
D	4	7

Livre: 1

Inseguro

- Solicitações de crédito são realizadas de tempo em tempo;
- * B é atendido. Em seguida os outros fazem solicitação, ninguém poderia ser atendido;

Starvation

- É a situação onde um processo nunca consegue executar sua região crítica e, consequentemente, acessar o recurso compartilhado.

- Este problema ocorre quando dois ou mais processos esperam por um recurso alocado.

- No momento em que o recurso é liberado, o sistema deve determinar qual processo, entre os que estão esperando, ganhará acesso ao recurso. Caso essa escolha seja realizada de forma aleatória, existe a possibilidade de um processo nunca ser escolhido e sofrer *starvation.*

Starvation

- Uma outra implementação, que também pode gerar esse problema, é quando o sistema determina prioridades para os processos acessarem o recurso.

- A baixa prioridade de um processo em relação a outros, que concorram pelos mesmos recursos, pode levar o processo a sofrer *starvation*.

- Assim, sempre que o recurso é liberado, o sistema escolhe o processo mais prioritário, podendo ocorrer que processos de baixa prioridade esperem indefinidamente pelo recurso.

Starvation

- Uma solução bastante simples para esse problema é a criação de filas de pedidos de alocação para cada recurso;

- Sempre que um processo desejar um recurso, o pedido é colocado no final da fila associada a ele;

- Quando o recurso é liberado, o sistema seleciona o primeiro processo da fila. O esquema de o primeiro a chegar ser o primeiro a ser atendido (FIFO) elimina o problema de *starvation.*