

BCC722

Programação de Sistemas em Tempo Real

Deadlocks

Prof. Charles Garrocho

Deadlocks

- Considere os processos P1 e P2 ($S=Q=1$):

P1:

```
wait(S);  
wait(Q);  
...  
signal(S);  
signal(Q);
```

P2:

```
wait(Q);  
wait(S);  
...  
signal(Q);  
signal(S);
```

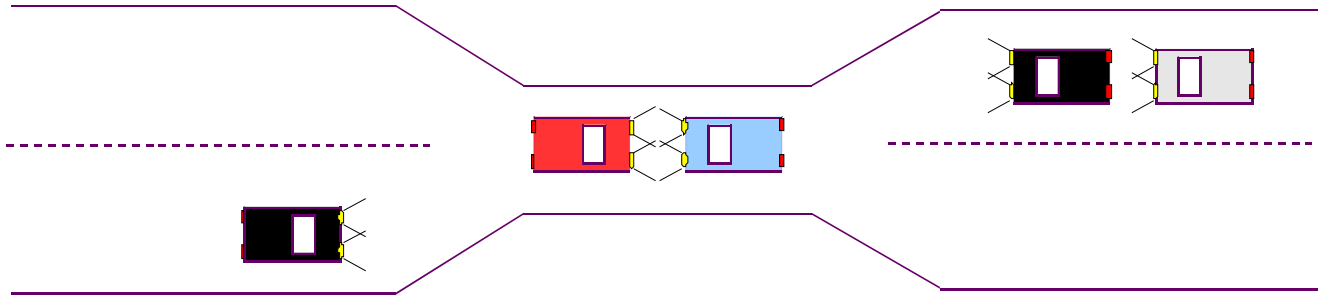
- Que tipo de problema pode ocorrer aqui?
 - Problema de aquisição e liberação de recursos
 - P1 e P2 ficam impedidos de prosseguir



O problema de *deadlock*

- Um conjunto de processos bloqueados, cada um de posse de um recurso e esperando por outro, já obtido por algum outro processo no conjunto
- Condições necessárias:
 - Exclusão mútua (Um processo acessa um recurso de cada vez)
 - Posse e espera (Um processo acessa um recurso e aguarda por outro já em posse)
 - Não-preempção (Recurso só é liberado após completar sua tarefa)
 - Espera circular (P0 aguarda P1, P1 aguarda P0)

Travessia de uma ponte estreita



- Cada parte da ponte é vista como um recurso
- Se um bloqueio ocorre (deadlock) pode ser resolvido com um carro dando ré
 - Carro libera recursos e retrocede
 - Vários carros podem ter que fazê-lo
- Inanição é possível

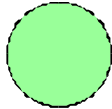
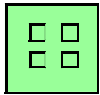
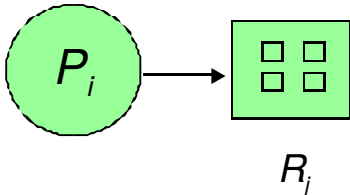
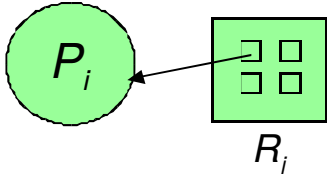
Modelo do sistema

- Recursos têm vários tipos R_1, R_2, \dots, R_m
 - Espaço de memória, disp. E/S
- Processos acessam recursos da mesma forma:
 - requisita
 - utiliza
 - libera

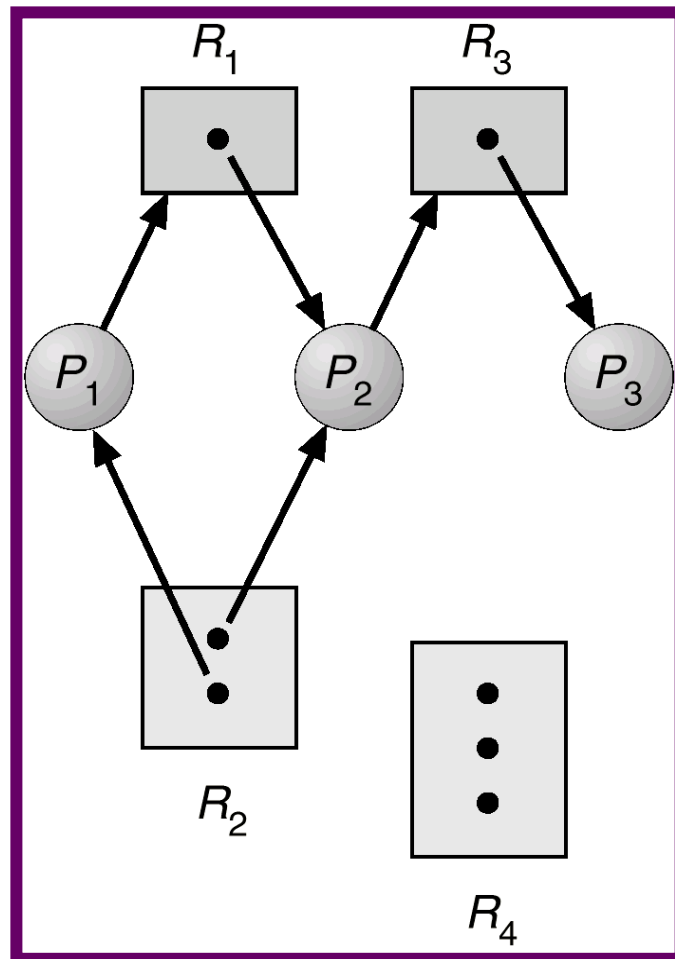
Grafo de alocação de recursos

- Vértices são divididos em dois tipos:
 - $P = \{P_1, P_2, \dots, P_n\}$, os processos no sistema
 - $R = \{R_1, R_2, \dots, R_m\}$, os recursos do sistema
- Arestas também são de dois tipos:
 - solicitação: aresta direcionada $P_i \rightarrow R_j$
 - atribuição: aresta direcionada $R_i \rightarrow P_j$

Grafo de alocação de recursos

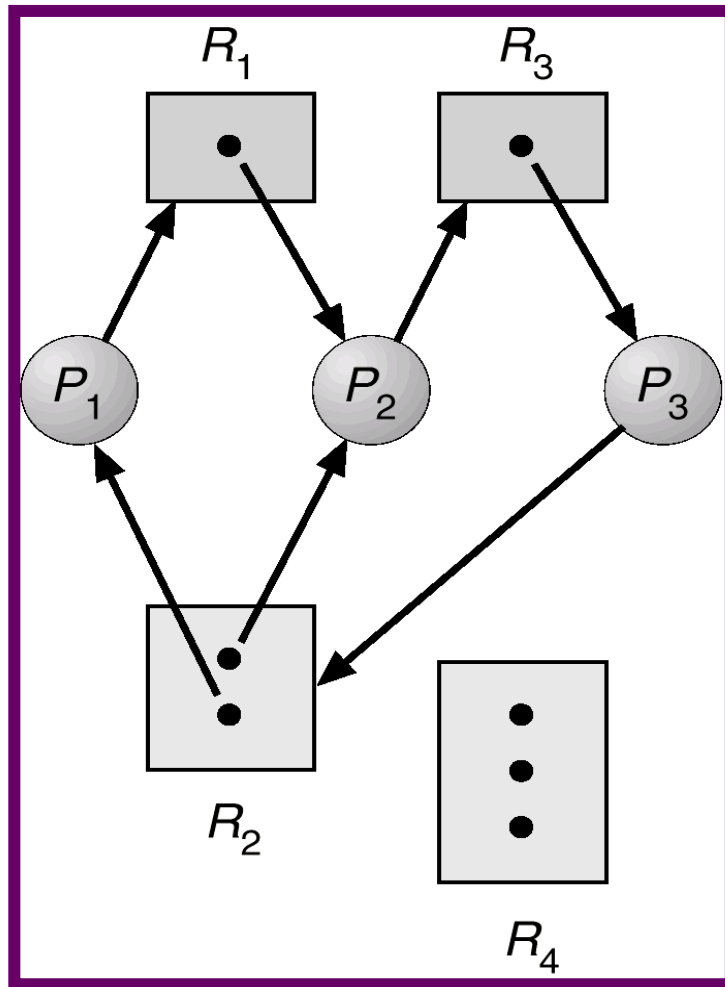
- Um processo: 
- Tipo de recurso com 4 instâncias 
- P_i requisita instância de R_j 
- P_i detém uma instância de R_j 

Grafo de alocação de recursos



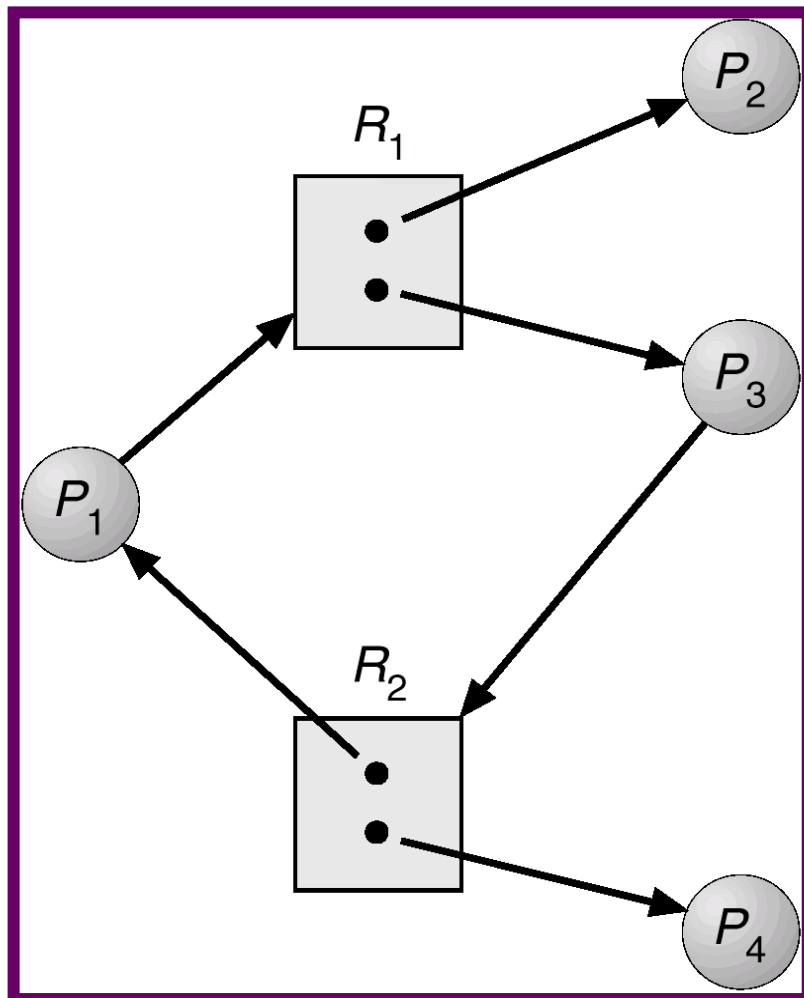
Não há deadlock

Grafo de alocação de recursos



Há deadlock

Grafo de alocação de recursos



Há ciclos, mas não há deadlock

Fatos básicos

- Se não há ciclos no grafo → não há *deadlock*
- Se o grafo contém ciclos
 - Se recursos só têm uma instância → *deadlock*
 - Se há mais de uma instância → possível *deadlock*

Formas de lidar com *deadlocks*

- Garanta que por construção que eles não acontecem
- Evite *deadlocks* antes que eles ocorram
- Detecte quando um *deadlock* ocorre e recupere o sistema a um estado aceitável
- Ignore o problema e faça de conta que eles nunca acontecem
 - Usado na maioria dos sistemas, inclusive o Linux!

Prevenção de *deadlocks*

- Garanta que pelo menos uma das condições originais para *deadlocks* nunca ocorra
 - Exclusão mútua
 - Posse durante a espera
 - Não preempção
 - Espera circular

Prevenção de *deadlocks*

- Garanta que pelo menos uma das condições originais para *deadlocks* nunca ocorra (1)
- Exclusão mútua
 - não há como evitar: necessária para recursos compartilhados

Prevenção de *deadlocks*

- Garanta que pelo menos uma das condições originais para *deadlocks* nunca ocorra (2)
- Posse e espera
 - Não permita que processos peçam recursos aos poucos:
 - pedem todos de uma vez ou
 - liberam todos os que detêm antes de pedir outros
 - Pode levar à baixa utilização dos recursos ou inanição

Prevenção de *deadlocks*

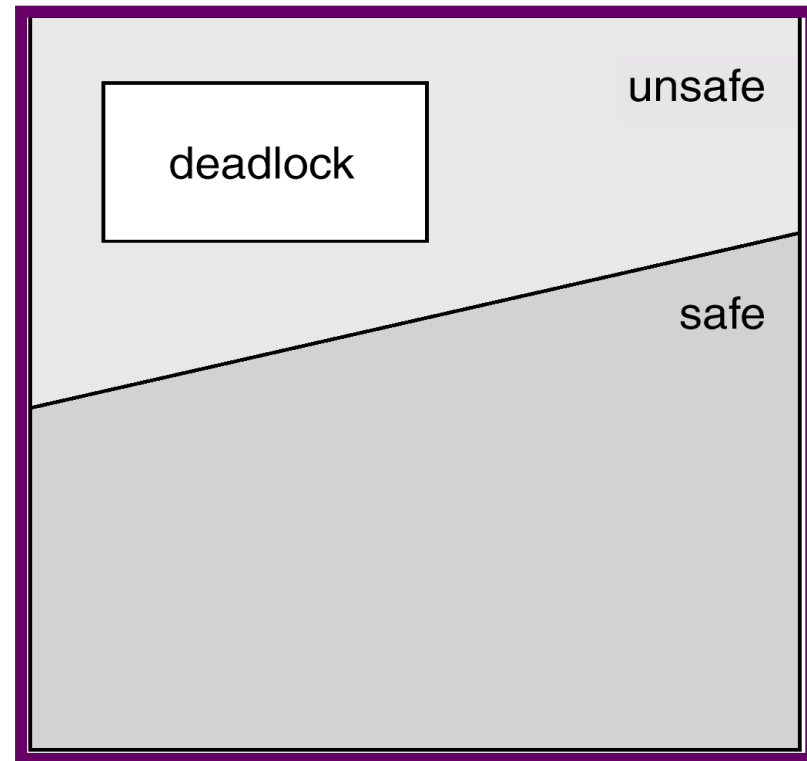
- Garanta que pelo menos uma das condições originais para *deadlocks* nunca ocorra (3)
- Não preempção
 - Se um processo não conseguir alocar um novo recurso, deve abrir mão dos que já detém
 - Implicitamente, eles serão adicionados à sua lista de requisições atual
 - O processo só poderá continuar quando todos os recursos puderem ser obtidos
 - Pode levar à inanição

Prevenção de *deadlocks*

- Garanta que pelo menos uma das condições originais para *deadlocks* nunca ocorra (4)
- Espera circular
 - Defina uma ordenação total para todos os tipos de recursos disponíveis
 - Exija que todo processo requisiite recursos sempre em ordem crescente
 - Exija que sempre que um processo que solicite um instancia do tipo R_j , ele tenha liberado quaisquer recursos R_i tal que $F(R_i) \geq F(R_j)$

Conceitos básicos

- Se um estado é seguro
 - Não há *deadlocks*
- Se um estado é inseguro
 - Há possibilidade de *deadlocks*
- Impedimento:
 - Garantir que o sistema nunca entre um estado inseguro



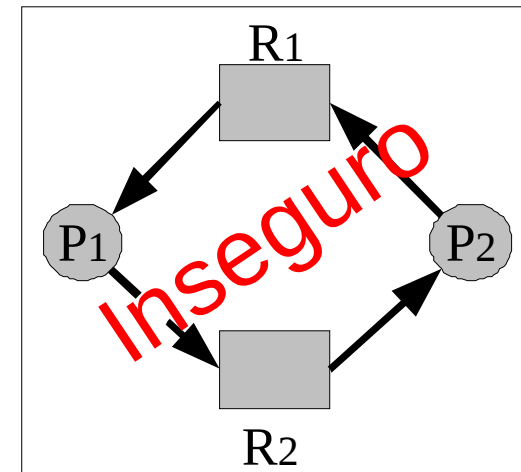
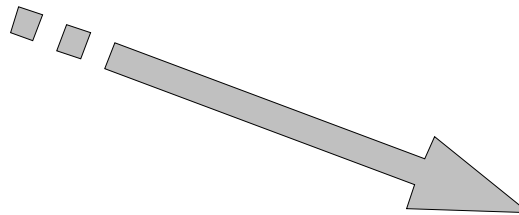
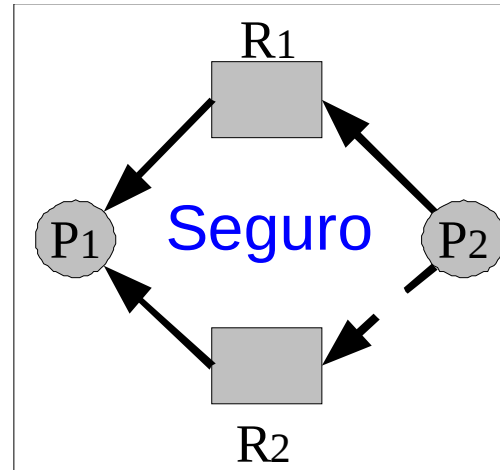
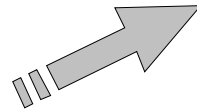
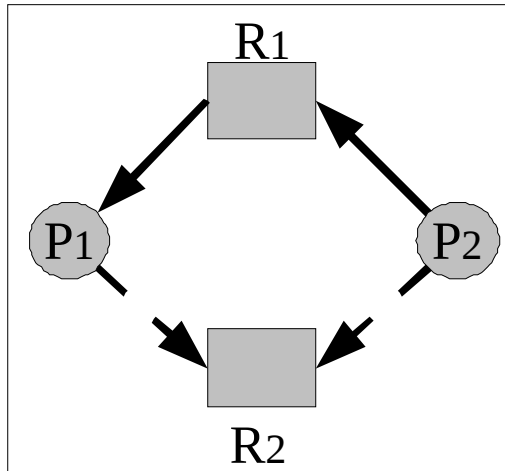
Algoritmo do grafo de alocação de recursos

- Aresta de requisição $P_i \rightarrow R_j$ (tracejada)
 - Processo P_i pode vir a requisitar R_j
 - Torna-se aresta de solicitação quando o pedido ocorre realmente
 - Quando o recurso é liberado aresta volta a ser de requisição
- Todos os recursos devem ser requisitados *a priori* no sistema

Algoritmo do grafo de alocação de recursos

- Antes de atender qualquer requisição:
 - Verifica se ao atender a requisição (inverter a aresta de requisição) cria-se um ciclo
 - Se o ciclo existe, estado seria inseguro
 - Processo é suspenso temporariamente
 - Se o ciclo não existe, estado é seguro
 - Solicitação pode ser atendida

Algoritmo do grafo de alocação de recursos



Recuperação de *deadlocks*

- É preciso quebrar os ciclos no grafo
- Abortar um ou mais processos
 - Processo termina com erro
 - Estado do sistema pode ficar inconsistente
- Fazer a preempção de recursos
 - Processos que sofrem preempção precisam “retroceder” (*roll-back*) para um ponto anterior

Recuperação de *deadlocks*: questões

- Como escolher o(s) processo(s) vítima(s)?
- Como distribuir os recursos reclamados?
- Como evitar a inanição?