

Sistemas Operacionais 1

Deadlock

Robson Siscoutto
robson@unoeste.br

Deadlocks

- Caracterização do deadlock
- Métodos para manipulação de deadlocks
- Prevenção de deadlocks
- Impedimento de deadlocks
- Detecção de deadlocks
- Recuperação de deadlocks
- Algoritmo do Banqueiro

Deadlocks em Sistemas Centralizados

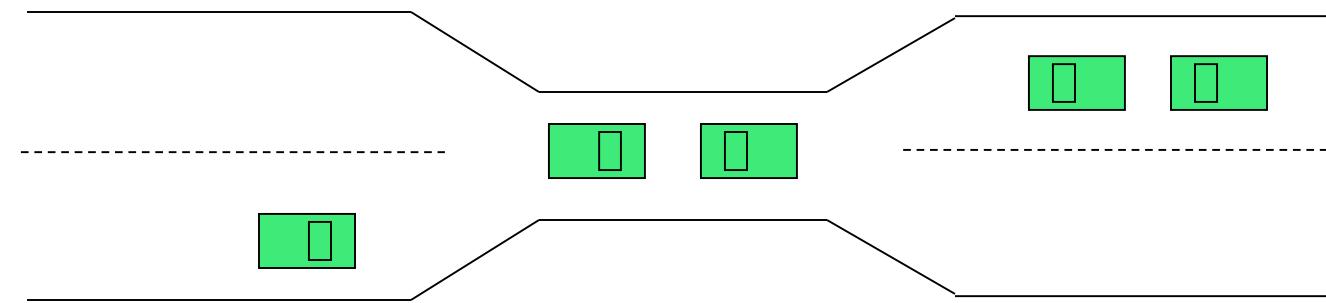
- Quando vários processos competem por um numero finito de recursos, poderá ocorrer o seguinte problema:
 - Um processo solicita um recurso e o recurso não está disponível no momento;
 - O processo entra no estado de espera – bloqueado;
 - Pode ficar neste estado indefinidamente, pois os recursos solicitados podem estar sendo mantidos por processos também bloqueados;
 - Essa situação é chamada de **DEADLOCK** ou Impasse;

O Problema de *Deadlocks*

- Conjunto de processos bloqueados cada um mantendo um recurso e esperando para adquirir um recurso, mantido por um outro processo no conjunto
- **Exemplo:**
 - Sistema com 2 *disk drives*
 - P_1 e P_2 cada um mantendo um *disk drive* e cada um necessitando de um outro
- **Exemplo:** semáforos A e B , iniciados com 1

P_0	P_1
wait(A);	wait(B)
wait(B);	wait(A)

Exemplo para Atravessar uma Ponte



- Tráfego somente em uma direção
- Cada seção da ponte pode ser vista como um recurso
- Se ocorre *deadlock*, pode ser resolvido se um carro voltar (**preempção de recursos** e **rollback**)
- Diversos carros podem ter que voltar se ocorre *deadlock*
- Pode causar ***starvation***

Deadlocks em Sistemas Centralizados

- Um sistema consiste de um numero finito de recursos a serem distribuídos entre os processos concorrentes;
- Um conjunto de processos está em deadlock quando todos os processos no conjunto estão esperando por um evento que pode ser causado apenas por outro processo no conjunto;
 - Eventos = requisição e liberação de recursos;
 - Exemplo:
 - Sistema possui dois tipos de dispositivos: impressora e unidade de fita.
 - P1 possui o recurso impressora e P2 possui o recurso unidade de fita;
 - Ambos necessitam o recurso do outro – DEADLOCK;

Deadlocks em Sistemas Centralizados

- Modelo do Sistema:
 - Tipos de Recursos R₁, R₂, . . . , R_n
 - Espaço de Memória, ciclos de CPU, dispositivos de I/O
 - Cada tipo de recurso R_i tem W_i instâncias.
 - Duas CPUs, então o tipo de recurso CPU terá duas instâncias;
 - Cada processo utiliza um recurso na seguinte seqüência:
 - **Requisição** – se não puder ser atendido, processo é bloqueado até poder obter o recurso
 - **Uso**
 - **Liberação**

Deadlocks em Sistemas Centralizados

- Caracterização de DEADLOCK:
 - Deadlock pode ocorrer se as quatro condições seguintes ocorrerem ao mesmo tempo:
 - **Exclusão Mutua:** somente um processo por vez pode usar um recurso.
 - **Posse e Espera:** um processo mantém pelo menos um recurso e esteja esperando para obter outros recursos adicionais mantidos por outros processos;

Deadlocks em Sistemas Centralizados

- Caracterização de DEADLOCK - quatro condições de Coffman :
 - **Não-preempção:** os recursos não podem sofrer preempção, ou seja, um recurso só pode ser liberado voluntariamente pelo processo que o mantém, depois que o processo tiver completado sua tarefa.
 - **Espera Circular:** deve haver um conjunto $\{P_0, P_1, \dots, P_n\}$ de processo em espera, de modo que:
 - P_0 esteja esperando por um recurso que é mantido por P_1 ,
 - P_1 esteja esperando por um recurso que é mantido por $P_2, \dots,$
 - P_{n-1} esteja esperando por um recurso que é mantido por P_n ,
 - P_n esteja esperando por um recurso que é mantido por P_0 .

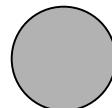
Deadlocks em Sistemas Centralizados

- **Grafo de Alocação de Recursos:**

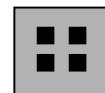
- Um conjunto de vértices **V** e um conjunto de arestas **A**.
- V é dividido em dois tipos diferentes de nós:
 - **P** = {P₁, P₂, ..., P_n}, o conjunto compostos de todos os **processos ativos** no sistema.
 - **R** = {R₁, R₂, ..., R_m}, o conjunto composto de todos os **tipos de recursos** no sistema.
- **Aresta de Pedido/Requisição** – aresta direcionada **P_i → R_j**
 - P_i solicitou e espera R_j
- **Aresta de Atribuição** – aresta direcionada **R_j → P_i**
 - R_j foi alocada para P_i

Deadlocks em Sistemas Centralizados

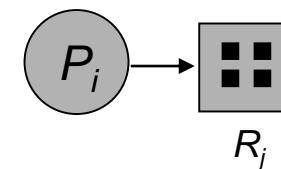
- Processo



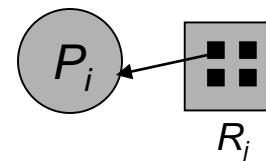
- Tipo de Recurso com 4 instâncias



- P_i requisita instância de R_j



- P_i está mantendo uma instância de R_j



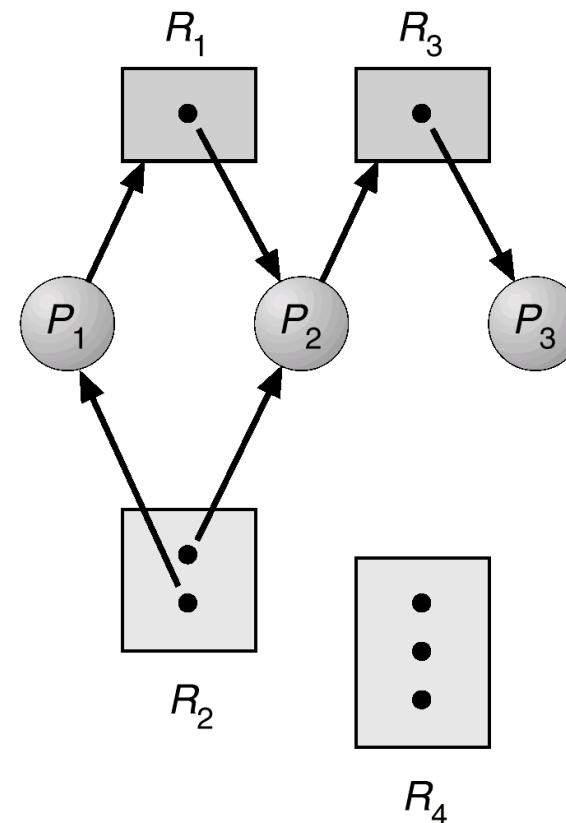
Deadlocks em Sistemas Centralizados

- Exemplo **sem Deadlock** de um Grafo de Alocação:

- 3 Conjuntos: P, R e A
 - $P = \{P_1, P_2, P_3\}$; $R = \{R_1, R_2, R_3, R_4\}$
 - $A = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2,$
 $R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

- Instâncias de Recurso:
 - 1 instancia de R1, 2 instância de R2
 - 1 instancia de R3, 3 instancia de R4

- Estados do Processo:
 - P1 mantém instancia de R2 e espera por R1
 - P2 mantém instancia de R1 e R2 e espera por R3
 - P3 mantém instancia de R3

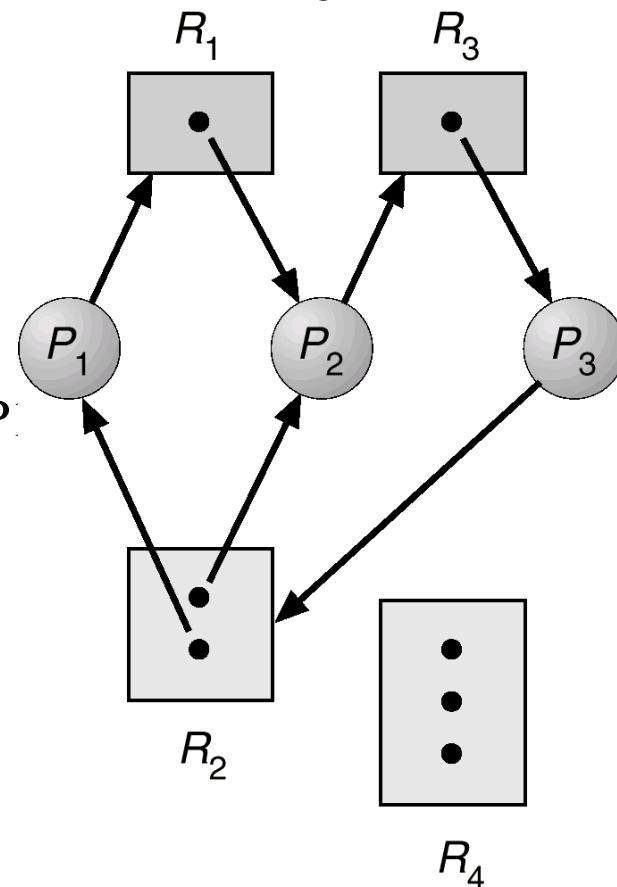


Deadlocks em Sistemas Centralizados

- Exemplo com Deadlock de um Grafo de Alocação:

- O grafo contém um ciclo,
logo existe um deadlock.

- Dois Ciclos:
 - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
 - $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



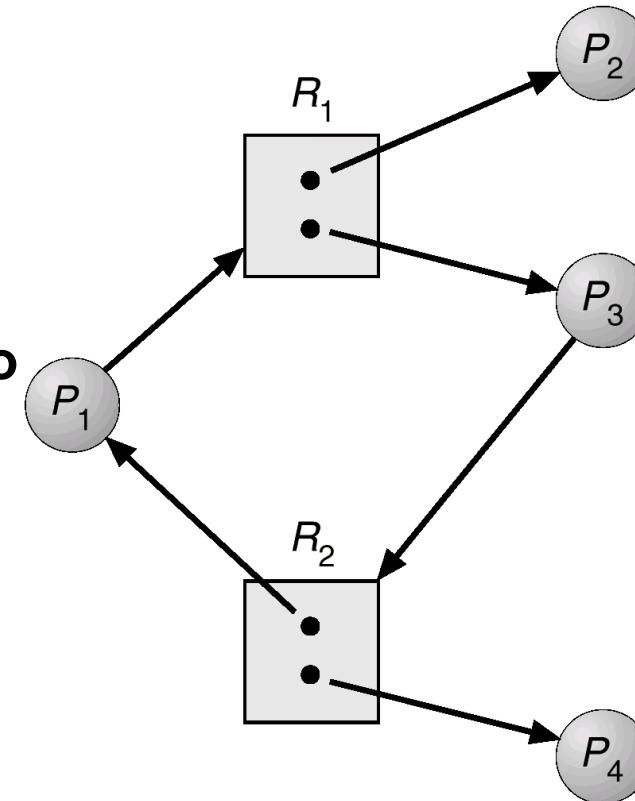
Deadlocks em Sistemas Centralizados

- Grafo de Alocação com **um ciclo mas não em Deadlock**

- Se P4 liberar sua instância de R2,
 - P3 terá a instancia solicitada,
 - O clico de Deadlock será quebrado.

- Se um **grafo de alocação não tiver Ciclo**
 - O Sistema não estará em Deadlock

- Se um **grafo tiver um ciclo**
 - Se somente uma instancia por tipo de recurso, então Deadlock;
 - Se varias instancias por tipo de recurso possibilidade de Deadlock



Deadlocks em Sistemas Centralizados

- Três Métodos para **Tratar Deadlocks**:
 1. Protocolo para garantir que o sistema **nunca** entre em estado de deadlock.
 - a) **Prevenir deadlocks**: métodos para garantir que pelo menos uma das 4 condições necessárias para o deadlock não venha ocorrer;
 - b) **Impedir deadlocks**: SO deve receber informação adicional previamente sobre recursos que processos irão solicitar e utilizar;
 2. Permitir que o sistema **entre em estado de deadlock** e, em seguida, se recupere (**detecção e Recuperação**);
 3. **Ignorar o problema e fingir** que os deadlocks nunca ocorrem no sistema (também conhecido como **Algoritmo do Avestruz**);
 - usado por muitos sistemas operacionais, incluindo UNIX;



Deadlocks em Sistemas Centralizados

Prevenção e Impedimentos de Deadlocks;

Deadlocks em Sistemas Centralizados

- Para Garantir que os Deadlocks nunca ocorrerão:
 - Prevenção e Impedimentos de Deadlocks;
 - Prevenção de Deadlocks:
 - Conjunto de métodos utilizados para garantir que pelo menos uma das condições necessários não seja válida.
 - Os métodos previnem Deadlock **limitando a maneira como os pedidos de recursos** podem ser feitos:
 - Exclusão Mútua – não valido para recursos compartilhados; deve ser valido para recursos não compartilhados.
 - Ex: arquivos apenas de leitura, os processos poderão ter acesso simultaneamente
 - Ex: impressa: não pode ter acesso ao mesmo tempo por vários processos;

Deadlocks em Sistemas Centralizados

- **Prevenção de Deadlocks**

- Os métodos previnem Deadlock limitando a maneira como os pedidos de recursos podem ser feitos:
 - **Posse e Espera** – deve garantir que sempre que um processo solicitar um recurso, ele não mantenha outros recursos. Dois Protocolos:
 1. Requer que o processo solicite e aloque todos os seus recursos antes de começar a execução,
 2. Permite que um processo solicite recursos apenas quando o processo não tiver nenhum recurso.

Deadlocks em Sistemas Centralizados

- **Prevenção de Deadlocks**

- **Posse e Espera** – Desvantagens do Protocolos:

- **Baixa utilização dos Recursos:**

- Muitos recursos podem se alocados mas não utilizados durante longos períodos;

- **Possibilidade de Starvation - paralisação:**

- Um processo que precisa de muitos recursos populares pode ter de esperar indefinidamente, porque pelo menos um dos recursos que necessita já está alocado a algum outro processo.

Deadlocks em Sistemas Centralizados

- **Prevenção de Deadlocks**

- **Não Preempção:**

- Se um processo que estiver de posse de alguns recursos solicitar outro recurso que não pode ser imediatamente alocado a ele, então todos os recursos sendo mantidos no momento são **liberados**.
 - Os recursos que foram liberados são adicionados a lista de recursos pelo quais o processo está esperando.
 - O processo será reiniciado quando puder obter novamente seus antigos recursos, assim como os novos que estão sendo solicitados.

Deadlocks em Sistemas Centralizados

- **Prevenção de Deadlocks**

- **Espera Circular:**

- Impor uma ordem total sobre todos os tipos de recursos e exigir que cada processo solicite recursos em ordem crescente de enumeração.
 - Um processo poderá solicitar inicialmente qualquer numero de instancias de um tipo de recurso, digamos R_i ;
 - Depois disso, o processo poderá solicitar instancias do tipo de recurso R_j , se e somente se $F(R_j) > F(R_i)$;
 - Se $F(R_i) \geq F(R_j)$ – deve liberar as instancias R_i

Deadlocks em Sistemas Centralizados

- **Impedimento (*avoidance*) de Deadlock:**

- Requer que o sistema operacional **recebe informações adicionais** relativas a quais recursos um processo solicitará e utilizará durante sua via útil:
 - Simplesmente este **modelo requer que cada processo declare o numero máximo de recursos** de cada tipo que pode precisar.
 - O algoritmo de prevenção de deadlock dinamicamente examina o estado dos recursos alocados para assegurar que nunca irá acontecer um deadlock - condições de espera circular.
 - O estado de alocação de recursos é definido pelo numero de recursos disponíveis e alocados e pela demanda máxima dos processos.

Deadlocks em Sistemas Centralizados

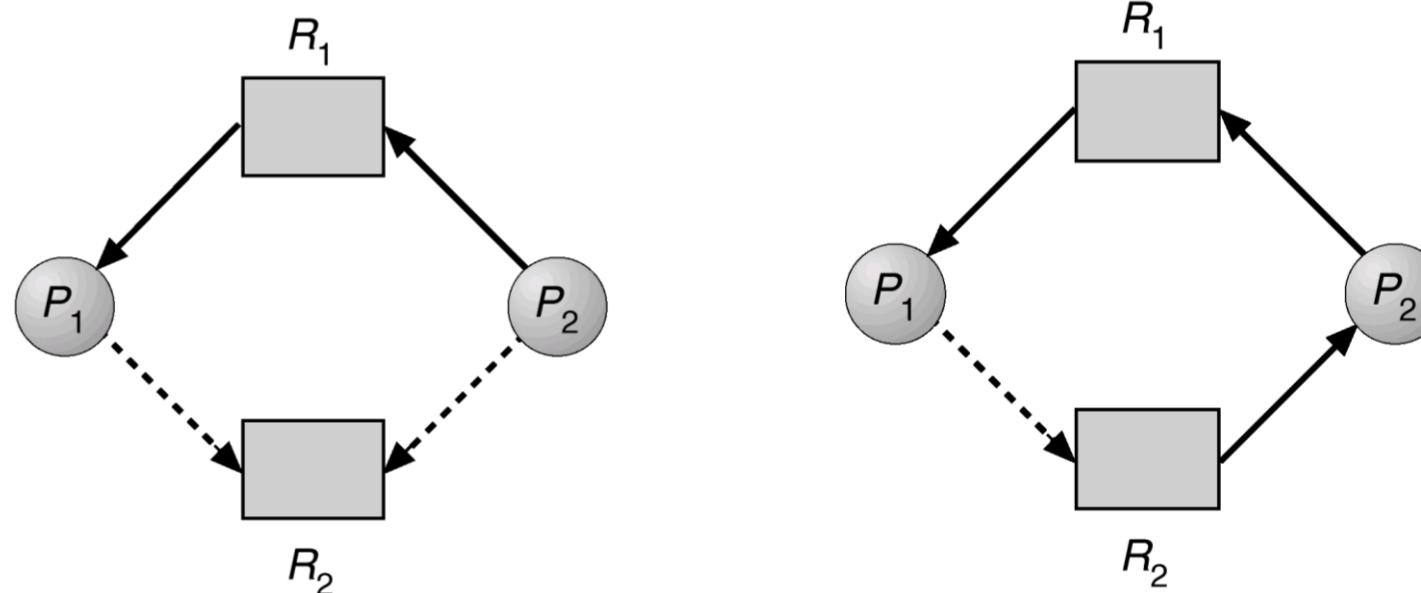
- **Impedimento de Deadlock:**
 - Além das arestas de pedido e atribuição:
 - Nova **Aresta de Demarcação** – representa por linha tracejada.
 - Uma aresta de demarcação $P_i \rightarrow R_j$:
 - Indica que o processo P_i pode solicitar o recurso R_j em algum momento no futuro;
 - A aresta de demarcação $P_i \rightarrow R_j$ é convertida para aresta de pedido;
 - Quando o recurso é liberado, a aresta de atribuição $R_j \rightarrow P_i$ é convertida em aresta de demarcação $P_i \rightarrow R_j$;
 - Os recursos devem ser reivindicados *a priori* no sistema.

Deadlocks em Sistemas Centralizados

- **Impedimento de Deadlock:**
 - Vamos supor que P_i solicite o recurso R_j :
 - O pedido só poderá ser concedido se a conversão da aresta de pedido $P_i \rightarrow R_j$ em uma aresta de atribuição $R_j \rightarrow P_i$, **não resulte na formação de um ciclo no grafo** de alocação de recursos;
 - **Se não existirem ciclos**, a alocação do recurso deixara o sistema em **estado seguro**.
 - Se um **ciclo for encontrado**, a alocação colocará o sistema em estado inseguro. Portanto, **P_i terá que esperar** que seus pedidos sejam satisfeitos;

Deadlocks em Sistemas Centralizados

- **Impedimento de Deadlock:**
 - Vamos supor que P2 solicite o recurso R2:
 - Embora R2 esteja livre no momento (figura a), não podemos aloca-lo para P2, já que essa ação criara um ciclo no grafo (figura b).



Deadlocks em Sistemas Centralizados

Detecção de Deadlock

Deadlocks em Sistemas Centralizados

- **Detecção de Deadlock:**

- Se um determinado sistema não emprega **algoritmo de prevenção ou impedimento de deadlock**, poderá ocorrer um deadlock.
- Neste caso, o sistema deverá fornecer umas das seguintes opções:
 1. Um **algoritmo que examine** o estado do sistema para determinar se ocorreu um deadlock;
 2. Um **algoritmo para recuperar** o sistema do deadlock;

Deadlocks em Sistemas Centralizados

- **Detecção de Deadlock:**

- Pode-se definir um algoritmo de detecção de deadlocks que utilize uma variante do grafo de alocação de recursos, chamado **grafo de espera**.
 - Nós são processos.
 - **P_i → P_j**, P_i está esperando que **P_j libere um recurso que P_i necessita**.
 - Uma aresta de **P_i → P_j** existe em um grafo de espera se e somente se o grafo de alocação de recursos correspondente contiver duas arestas **P_i → R_q** e **R_q → P_j**

Deadlocks em Sistemas Centralizados

- **Detecção de Deadlock:**

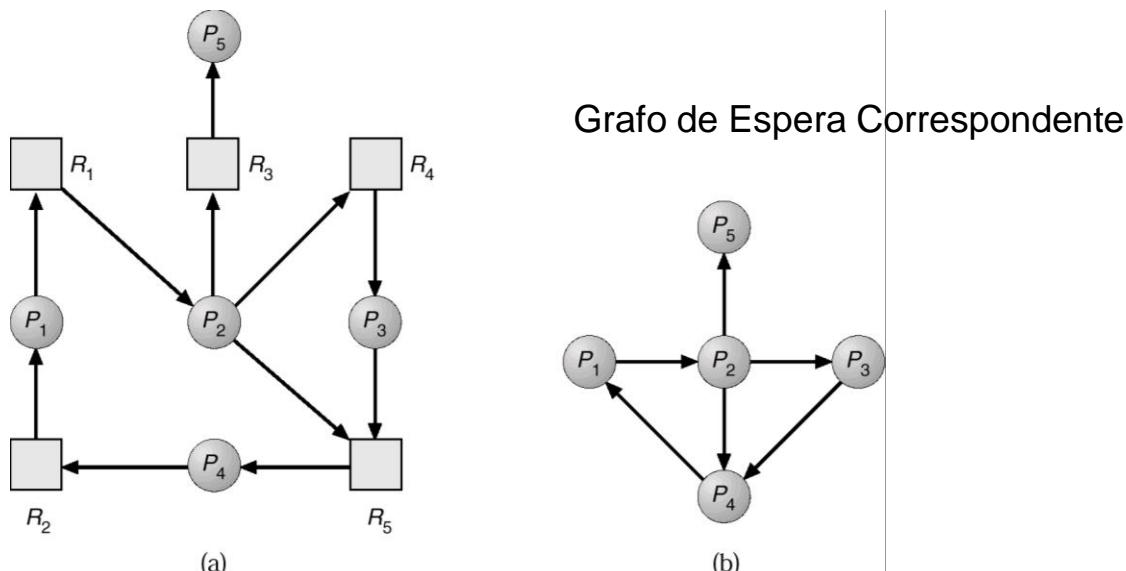
- **Grafo de Espera:**

- Para manter o grafo de espera atualizado, o sistema chama periodicamente o algoritmo que busca por ciclos no grafo de alocação de recursos.
 - Um algoritmo para detectar um ciclo no grafo requer na ordem de n^2 operações, onde **n** é o numero de vértices no grafo.

Deadlocks em Sistemas Centralizados

- **Detecção de Deadlock:**

- **Grafo de Espera:** Uma aresta de $P_i \rightarrow P_j$ existe em um grafo de espera se e somente se o grafo de alocação de recursos correspondente contiver duas arestas $P_i \rightarrow R_q$ e $R_q \rightarrow P_j$



Deadlocks em Sistemas Centralizados

Recuperação de Deadlock

Deadlocks em Sistemas Centralizados

- **Recuperação de Deadlock – Término do Processo:**
 - **Dois métodos utilizados:**
 1. Abortar todos os processos em deadlocks:
 - Pode ter um custo alto no caso de liberar um processo com um tempo longo já processado;
 2. Abortar um processo de cada vez até eliminar o ciclo de deadlock:
 - Alto custo, pois depois de abortar cada processo, é chamado o algoritmo de detecção de deadlock para determinar se há ainda deadlocks.

Deadlocks em Sistemas Centralizados

- **Recuperação de Deadlock – Término do Processo :**
 - Em qual ordem deve ser escolhido o processo a ser abortado:
 - Prioridade dos processos;.
 - Quanto tempo ficou computando e por quanto tempo ele ainda continuaria executando até completar;
 - Quantos e que tipos de recursos utilizou;
 - Quantos recursos mais o processo precisa para terminar;
 - Quantos processos precisarão ser terminados.
 - Se o processo é interativo ou Batch?

Deadlocks em Sistemas Centralizados

- **Recuperação de Deadlock – Preempção de Recursos:**
 - Para eliminar deadlocks usando preempção de recursos, fazemos a preempção sucessiva de alguns recursos dos processos e conferimos esses recursos a outros processos até o ciclo de deadlock seja quebrado:
 - **Seleção de uma vítima** – fatores de custo mínimo.
 - **Rollback** (volta ao passado ou retrocesso) – retornar o processo para um estado seguro e reinicia-lo a partir desse estado.
 - **Starvation** (paralisação) – escolher sempre a mesma vítima, incluindo o numero do rollback no fator de custo.

Deadlocks em Sistemas Centralizados

- **Considerações sobre Deadlocks:**

- Combinar as três técnicas
 - Prevenção
 - Impedimentos
 - Detecção
- Permite uso da melhor técnica para cada de recursos no sistema.
- Dividir os recursos em classe ordenadas hierarquicamente
- Use da técnica mais apropriada por controlar paralisações completas dentro de cada classe.

Banker's Algorithm

- Múltiplas instâncias dos recursos;
- Cada processo deve *a priori* declarar a sua demanda de uso máxima
- Quando um processo solicita um recurso ele pode ter que esperar
- Quando um processo consegue todos os seus recursos, ele deve liberá-los numa quantidade de tempo finita

Estrutura de Dados: *Banker's Algorithm*

Suponha

n = número de processos,

m = número de tipos de recursos

- **Available**: Vetor de tamanho m . Se $\text{Available}[j] = k$, existem k instâncias dos recursos do tipo R_j disponíveis
- **Max**: matriz $n \times m$. Se $\text{Max}[i,j] = k$, então processo P_i pode pedir até k instâncias do recurso do tipo R_j

Estrutura de Dados: *Banker's Algorithm*

- **Allocation**: matriz $n \times m$. Se $\text{Allocation}[i,j] = k$ então P_i está alocando k instâncias de R_j
- **Need**: matriz $n \times m$. Se $\text{Need}[i,j] = k$, então P_i pode necessitar de mais k instâncias de R_j para terminar sua tarefa

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$$

Algoritmo Seguro

1. Deixe **Work** e **Finish** serem vetores de tamanho **m** e **n**, respectivamente. Inicie:

Work = Available

Finish [i] = false for $i = 0, 1, \dots, n - 1$

2. Encontre um i tal que ambos:

(a) **Finish [i] = false**

(b) **Need_i ≤ Work**

Se não existe tal i , vá para o passo 4

3. **Work = Work + Allocation_i**

Finish[i] = true

vá para o passo 2

4. Se **Finish [i] == true** para todo i , então o sistema está em um **estado seguro**

Além disso, se **Finish[i] == false**, então P_i está em **deadlock**

Algoritmo requer operações da ordem de $O(m \times n^2)$ para detectar se o sistema está em estado de *deadlocked*

Algoritmo de Pedido de Recurso por Processo P_i

Request = vetor de pedido para o processo P_i . If $\text{Request}_i[j] = k$ então processo P_i quer k instâncias do recurso de tipo R_j

1. Se $\text{Request}_i \leq \text{Need}_i$, vá para o passo 2. Caso contrário, condição de erro, uma vez que o processo excedeu sua demanda máxima
2. Se $\text{Request}_i \leq \text{Available}$, vá para o passo 3. Caso contrário P_i deve esperar, uma vez que os recursos não estão disponíveis

Algoritmo de Pedido de Recurso por Processo P_i

3. Pretende alocar recursos pedidos para P_i modificando o estado da seguinte forma:

$$\mathbf{Available} = \mathbf{Available} - \mathbf{Request};$$

$$\mathbf{Allocation}_i = \mathbf{Allocation}_i + \mathbf{Request}_i;$$

$$\mathbf{Need}_i = \mathbf{Need}_i - \mathbf{Request}_i;$$

Se $\text{safe} \Rightarrow$ os recursos são alocados a P_i

Se $\text{unsafe} \Rightarrow P_i$ deve esperar e o estado de alocação antigo é restaurado

Exemplo 1 do Algoritmo do Banqueiro

- 5 Processos P_0 até P_4 ;
- 3 tipos de recursos: A (10), B (5), e C (7) instâncias
- *Snapshot* do tempo T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Existe alguma sequencia de processos onde consigo executar todos de forma segura?

Exemplo 1 do Algoritmo do Banqueiro

- O conteúdo da matriz **Need** é definido como **Max-Allocation**

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

- O sistema está em um estado seguro, porque a sequência $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfaiz o critério de segurança

Exemplo 2: P_1 solicita (1,0,2)

- Verifique que $\text{Request} \leq \text{Available}$ (que é, $(1,0,2) \leq (3,3,2) \Rightarrow \text{true}$)

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>	<u>Need</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 1	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

- Execução do algoritmo de segurança mostra que a sequência $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfaz a necessidade de segurança.

Exemplo 3:

- O pedido para **(3,3,0)** pelo **P_4** pode ser garantido?
- O pedido para **(0,2,0)** pelo **P_0** pode ser garantido?
- 5 Processos **P_0** até **P_4** ;
- 3 tipos de recursos: A (10), B (5), e C (7) instâncias
- *Snapshot* do tempo **T_0** :

	<u>Allocation</u>			<u>Max</u>	<u>Available</u>	<u>Need</u>
	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3