

## ***Banker's Algorithm***

- Múltiplas instâncias
- 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<sub>i</sub> ≤ Work**

Se não existe tal **i**, vá para o passo 4

3. **Work = Work + Allocation<sub>i</sub>**

**Finish[i] = true**

vá para o passo 2

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

# 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, raise error 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:

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

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

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

Se *safe*  $\Rightarrow$  os recursos são alocados a  $P_i$

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

# Exemplo 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>	
	<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

# Exemplo do Algoritmo do Banqueiro

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

	<u>Need</u>		
	A	B	C
$P_0$	7	4	3
$P_1$	1	2	2
$P_2$	6	0	0
$P_3$	0	1	1
$P_4$	4	3	1

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

## Exemplo: $P_i$ 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>Need</u>			<u>Available</u>		
	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>
$P_0$	0	1	0	7	4	3	2	3	0
$P_1$	3	0	2	0	2	0			
$P_2$	3	0	1	6	0	0			
$P_3$	2	1	1	0	1	1			
$P_4$	0	0	2	4	3	1			

## Exemplo: $P_i$ solicita (1,0,2)

- Execução do algoritmo de segurança mostra que a seqüência  $\langle P_1, P_3, P_4, P_0, P_2 \rangle$  satisfaçais a necessidade de segurança
- O pedido para **(3,3,0)** pelo  $P_4$  pode ser garantido?
- O pedido para **(0,2,0)** pelo  $P_0$  pode ser garantido?