

Engenharia de Computação

Arquitetura de Sistemas Operacionais

Prevenção e Tratamento de Deadlocks

Prof. Anderson Luiz Fernandes Perez
Prof. Martín Vigil

Sumário

- Introdução
- Caracterização do Deadlock
- Grafo de Alocação de Recursos
- Manipulação de Deadlock

Introdução

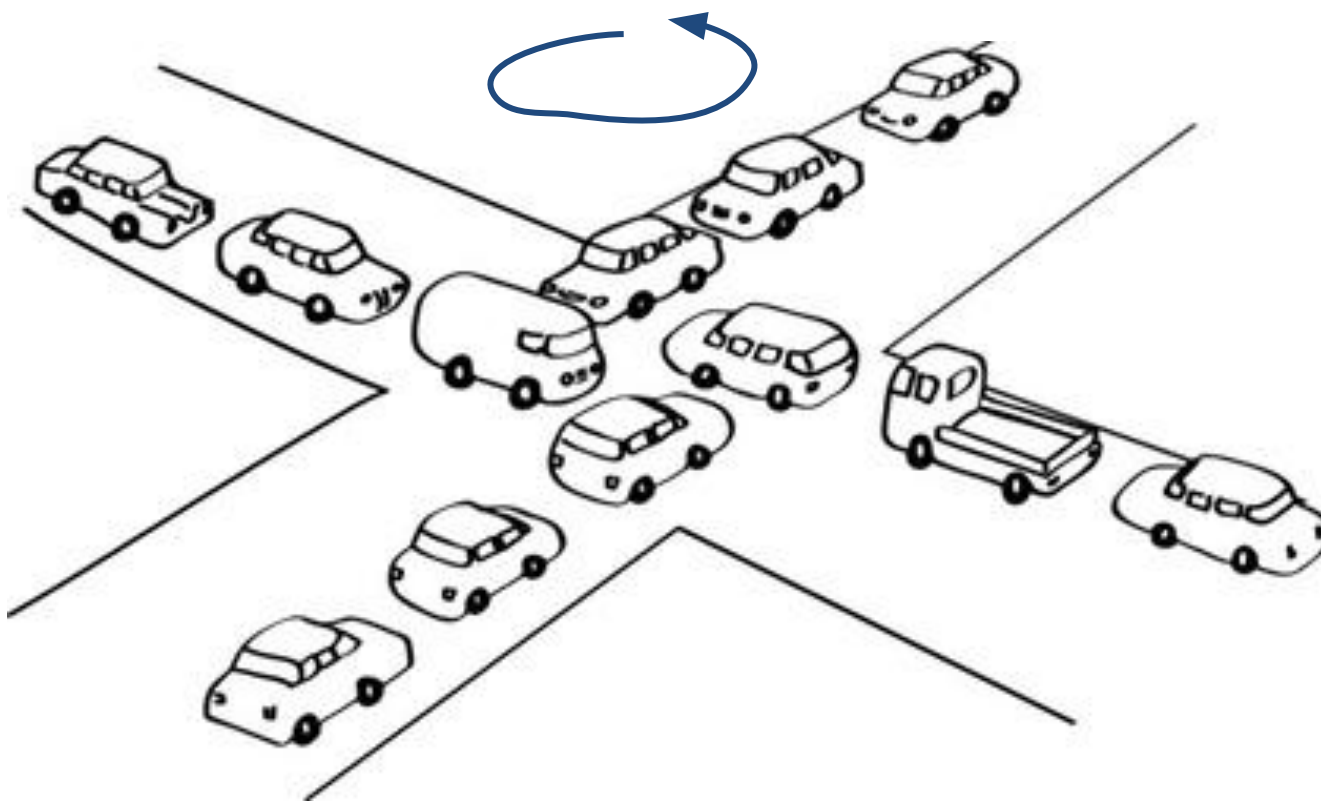
- Um processo deve solicitar o acesso a um recurso antes de utilizá-lo e liberá-lo após o uso.
- Um processo solicita recursos obedecendo a seguinte sequência:
 - **Solicitação**: o processo solicita o recurso. Se a solicitação não puder ser atendida imediatamente, então o processo solicitante deve esperar até poder adquirir o recurso.
 - **Uso**: o processo pode operar sobre o recurso.
 - **Liberação**: o processo libera o recurso.

Introdução

- Em sistemas multiprogramados os processos solicitam acesso a recursos finitos.
- Um processo pode ficar em **estado de espera** aguardando a **liberação** de um recurso que foi alocado por outro processo que também pode estar em **estado de espera**.

Introdução: exemplo real

- Espera circular



Introdução

- Um conjunto de processos está em **deadlock** se **cada processo** está **bloqueado esperando** um evento que somente pode ser causado por **outro processo também bloqueado** do conjunto.

Caracterização do Deadlock

- Em uma situação de deadlock os processos envolvidos **nunca** concluem sua execução e os recursos ficam **ocupados indeterminadamente**, impedindo que outros processos os utilizem.

Introdução

- A situação de deadlock só poderá ser alterada por alguma iniciativa que parta de um processo fora do conjunto dos processos.
- Deadlocks podem ocorrer quando vários processos recebem direitos de acesso exclusivo a dispositivos, arquivos, registros e etc.

Caracterização do Deadlock

- Uma situação de deadlock pode surgir se as quatro condições a seguir ocorrerem simultaneamente:
 1. **Exclusão Mútua:** em um determinado instante, cada recurso está em uma de duas situações:
 - associado a um único processo;
 - disponível.

Caracterização do Deadlock

2. **Posse e Espera:** processos que, em um determinado instante, retêm recursos concedidos anteriormente podem requisitar novos recursos.
3. **Não-Preempção:** recursos concedidos previamente a um processo não podem ser forçosamente tomados desse processo. Eles devem ser explicitamente liberados pelo processo que os retém.

Caracterização do Deadlock



4. **Espera Circular:** deve existir um encadeamento circular de dois ou mais processos; cada um deles encontra-se à espera de um recurso que está sendo usado pelo membro seguinte dessa cadeia.

Grafo de Alocação de Recursos

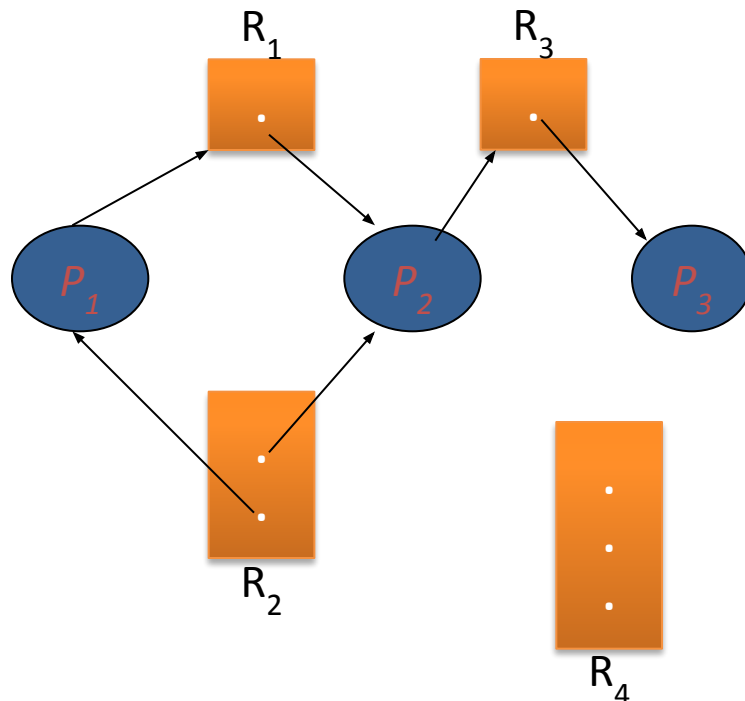
- É um grafo **orientado** que consiste em um conjunto de vértices V e um conjunto de arestas A .
- O conjunto de vértices V é **particionado** em dois tipos de nós diferentes:
 - Conjunto dos **processos**
 - $P = \{P1, P2, \dots, Pn\}$
 - Conjunto dos **recursos**
 - $R = \{R1, R2, \dots, Rn\}$

Grafo de Alocação de Recursos

- Uma aresta (i,j) direcionada do **processo P_i** **para o recurso R_j** é indicada por:
 - $P_i \rightarrow R_j$ (Aresta de requisição)
 - O processo **requisitou** o recurso.
- Uma aresta (i,j) direcionada do **recurso R_j** **para o processo P_i** é indicada por:
 - $R_j \rightarrow P_i$ (Aresta de atribuição)
 - Uma instância do recurso **está alocado** ao processo.

Grafo de Alocação de Recursos

- Exemplo:



Os conjuntos P, R e A:

$$P = \{P_1, P_2, P_3\}$$

$$R = \{R_1, R_2, R_3, R_4\}$$

$$A = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$$

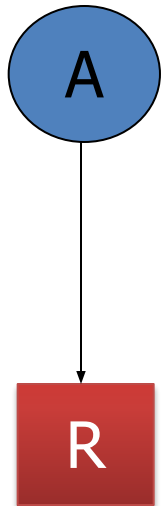
Grafo de Alocação de Recursos

- Se cada recurso tiver uma única instância, então um ciclo indica que ocorreu um deadlock.
- Se o ciclo envolver apenas um conjunto de recursos, cada qual com apenas uma única instância, então ocorreu um deadlock.
- Nesse caso, um ciclo no grafo é condição **necessária e suficiente** para a existência de deadlock.

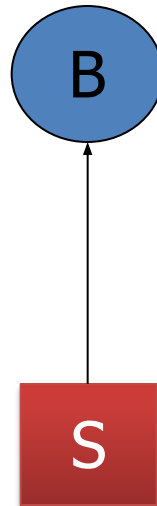
Grafo de Alocação de Recursos

- Diagramas de Alocação

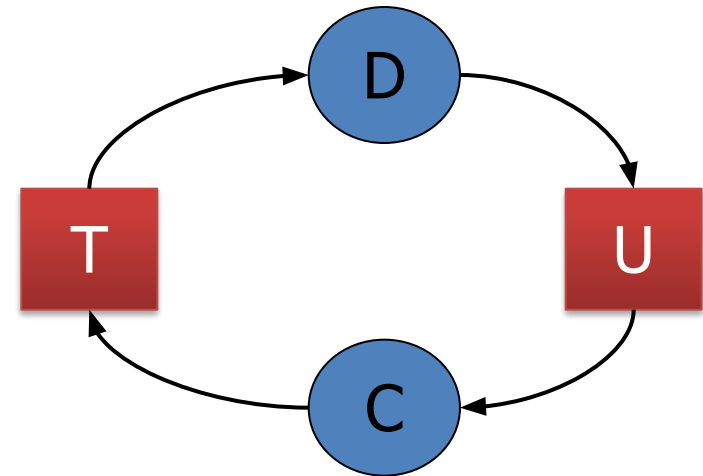
Processo A solicita
recurso R



Processo B retêm
recurso S



Deadlock (ciclo = C-T-D-U-C)



Grafo de Alocação de Recursos

- Se cada recurso tiver **várias instâncias**, então um ciclo **não significa** que ocorreu um deadlock.
- Nesse caso, um ciclo no grafo é uma condição necessária mas não **suficiente** para a existência de deadlock.

Manipulação de Deadlock



- O problema do deadlock pode ser lidado de uma entre três maneiras:
 - **Prevenir**: usa-se um **protocolo para prevenir** ocorrência de deadlocks, garantindo que o sistema nunca entrará em estado de deadlock.
 - **Remediar**: permitir que o sistema entre em um estado de deadlock, **detecte-o** e execute uma **recuperação**.
 - **Ignorar**: ignorar o problema e **assumindo** que deadlocks **nunca ocorrerão** no sistema.

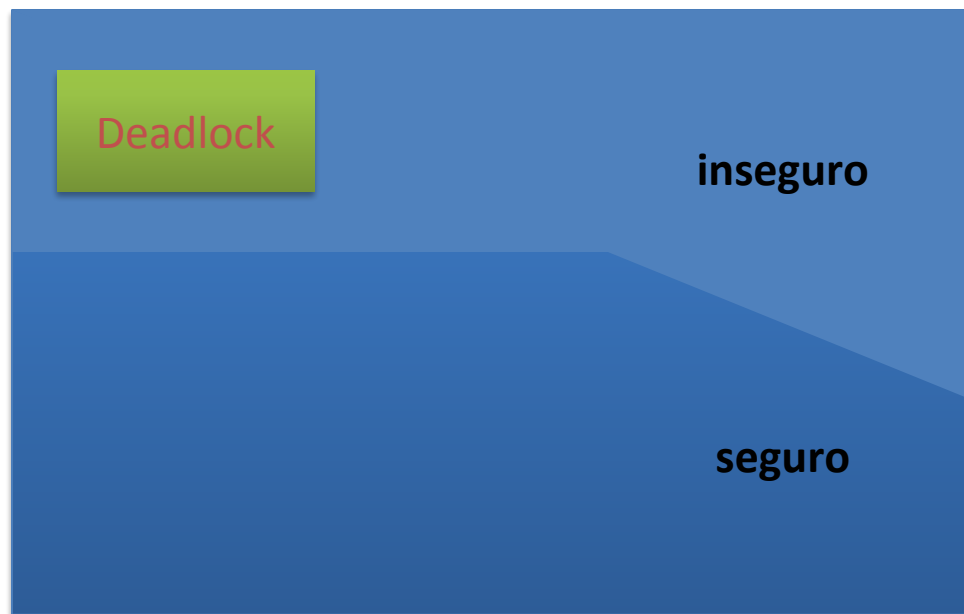
Manipulação de Deadlock



- Estados Seguros e Inseguros
 - Um estado é considerado seguro se ele **não está em situação de deadlock** e se **existe alguma ordem de escalonamento** na qual todo o processo possa ser executado **até a sua conclusão**, mesmo se, repentinamente, todos eles requisitem, de uma só vez, o máximo possível de recursos.

Manipulação de Deadlock

- Estados Seguros e Inseguros



Manipulação de Deadlock



- Estados Seguros e Inseguros
 - Exemplo:
 - 1 recurso com 12 instâncias
 - 3 processos: P_0 , P_1 e P_2
 - Processos estão usando 9 instâncias no total
 - Processos podem solicitar mais instâncias

Processo	Necessidades máximas	Necessidades correntes
P_0	10	5
P_1	4	2
P_2	9	2

Manipulação de Deadlock



- Estados Seguros e Inseguros

- **Exemplo:** recurso com 10 unidades => estados seguros?

Proc	Máx	Possui
A	9	3
B	4	2
C	7	2

Disponível = 3

Proc	Máx	Possui
A	9	3
B	4	4
C	7	2

Disponível = 1

Proc	Máx	Possui
A	9	3
B	-	0
C	7	2

Disponível = 5

t_1

t_2

t_3

tempo

Manipulação de Deadlock



- Estados Seguros e Inseguros

- **Exemplo:** recurso com 10 unidades => estados seguros?

Estados
seguros!

Proc	Máx	Possui
A	9	3
B	-	0
C	7	7

Disponível = 0

Proc	Max	Possui
A	9	3
B	-	0
C	-	0

Disponível = 7

t_4

t_5

tempo

Manipulação de Deadlock



- Estados Seguros e Inseguros

- **Exemplo:** recurso com 10 unidades=> estado seguro?

Proc	Máx	Possui
A	9	3
B	4	2
C	7	2

Disponível = 3

t_1

Proc	Máx	Possui
A	9	4
B	4	2
C	7	2

Disponível = 2

t_2

Proc	Máx	Possui
A	9	4
B	4	4
C	7	2

Disponível = 0

t_3

tempo

Manipulação de Deadlock



- Estados Seguros e Inseguros
 - **Exemplo:** recurso com 10 unidades (estado inseguro).

Estados
inseguros!

Proc	Máx	Possui
A	9	4
B	-	-
C	7	2

Disponível = 4

t_4

tempo

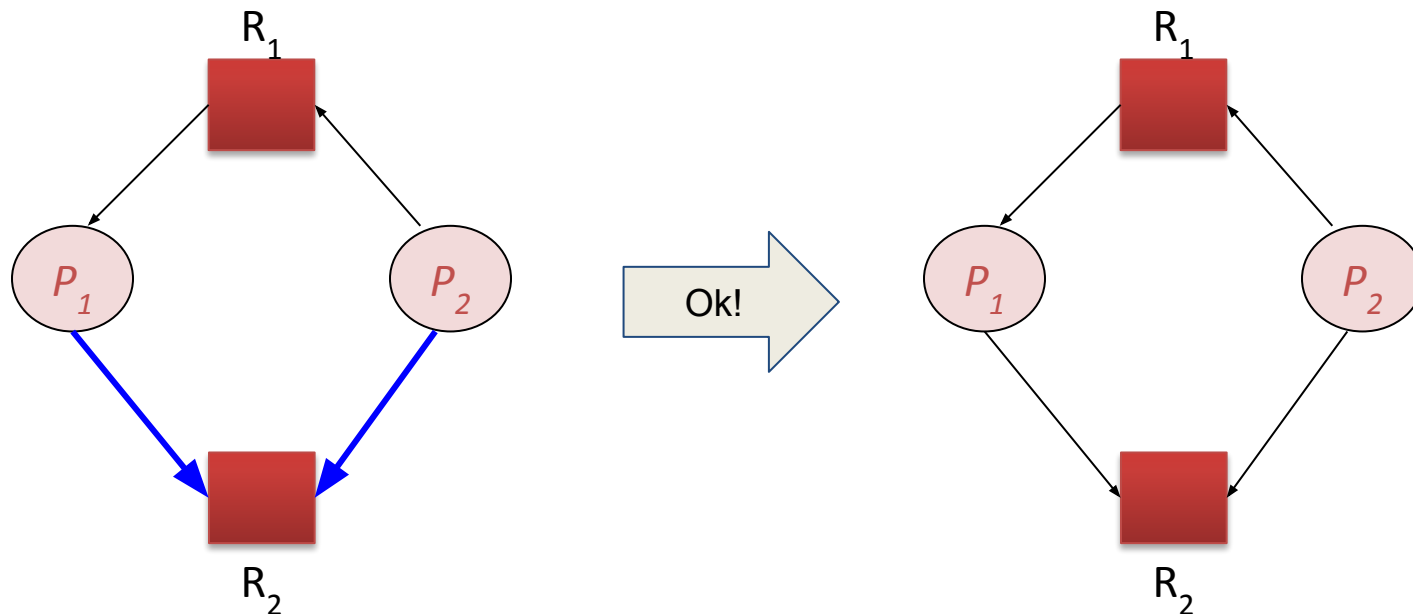
Manipulação de Deadlock



- Algoritmo do grafo de alocação de recursos
 - Não funciona qdo recurso tem múltipla instâncias
 - Cria-se um novo tipo de aresta chamada de aresta de pretensão (requisição).
 - Uma aresta de pretensão $P_i \rightarrow R_j$ indica que o processo P_i pode requisitar o recurso R_j em algum momento no futuro.
 - A requisição de um processo só poderá ser concedida se a conversão da aresta de requisição para uma aresta de atribuição não resultar na formação de um ciclo no grafo de alocação de recursos.

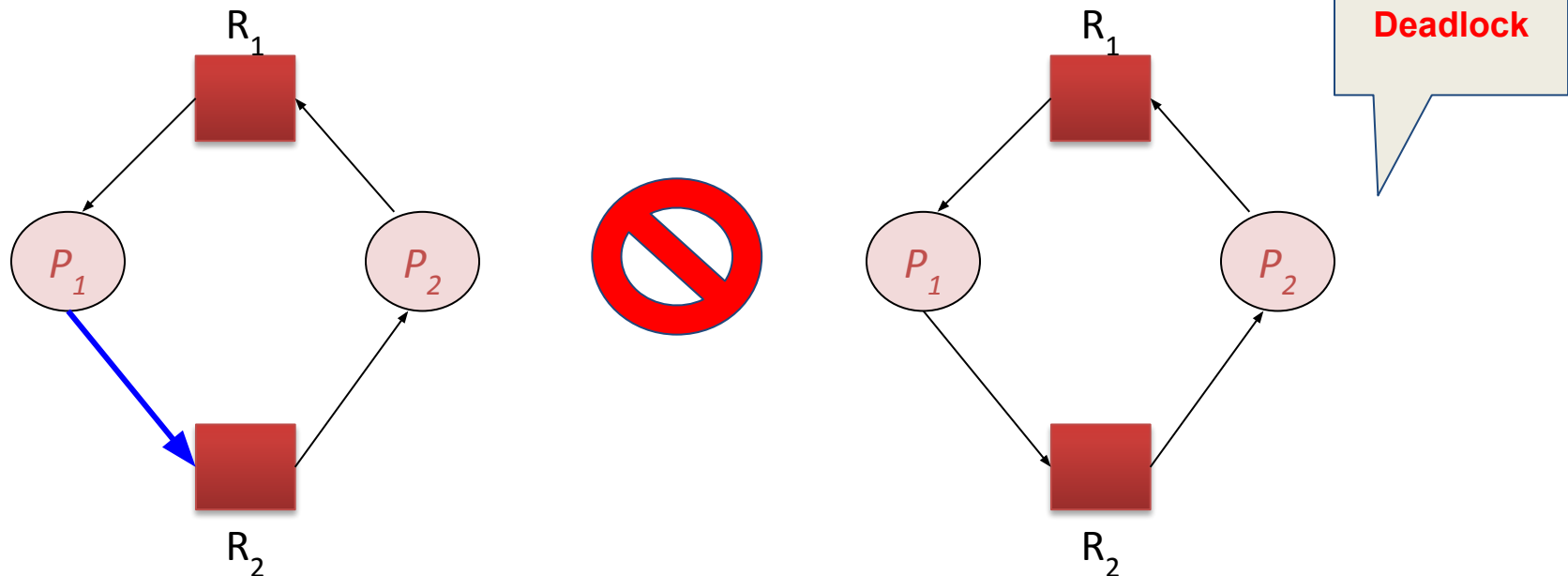
Manipulação de Deadlock

- Algoritmo do grafo de alocação de recursos
 - Exemplo (aresta de pretenção em **azul**)



Manipulação de Deadlock

- Algoritmo do grafo de alocação de recursos
 - Exemplo (aresta de pretenção em **azul**)



Manipulação de Deadlock



- Algoritmo do banqueiro
 - Útil em sistemas onde cada recursos possui múltiplas instâncias.
 - Quando um novo processo entra no sistema ele precisa declarar o número máximo de instâncias que pode precisar de cada recurso
 - Esse número não pode ultrapassar o número total de recursos do sistema.

Manipulação de Deadlock



- Algoritmo do banqueiro
 - O algoritmo do banqueiro precisa de algumas estruturas de dados para o estado do sistema.
 - Dois parâmetros a lembrar:

número de processos = **n**

número de tipos de recursos = **m**

Manipulação de Deadlock



- Algoritmo do banqueiro
 - **Disponível**: um vetor de tamanho m indica o número de recursos disponíveis de cada tipo. Se $Disponível[j]=k$, então há k instâncias disponíveis do recurso R_j .
 - **Máximo**: uma matriz $n \times m$ define a demanda máxima de cada processo. Se $Máximo[i][j]=k$, então o processo P_i pode requisitar no máximo k instâncias do recurso R_j .

Manipulação de Deadlock



- Algoritmo do banqueiro
 - **Alocação**: uma matriz $n \times m$ define a quantidade **atual** de recursos de cada tipo alocado a cada processo. Se $Alocação[i][j]=k$, então o processo P_i **está usando** k instâncias do recurso R_j .
 - **Necessário**: uma matriz $n \times m$ indica a necessidade restante de recursos para cada processo. Se $Necessário[i][j]=k$, então o processo P_i **pode vir a** precisar de mais k instâncias do recurso R_j .
 - **Obs.** $Necessário[i][j] = Máximo[i][j] - Alocação[i][j]$.

Manipulação de Deadlock



- Comparar vetores $A=\{a_1, \dots, a_n\}$ e $B=\{b_1, \dots, b_n\}$:

$$A \leq B \leftrightarrow \forall j [a_j < b_j]$$

Manipulação de Deadlock



- Algoritmo do banqueiro
 - Algoritmo da requisição de recursos
 - Seja Requisição_i o vetor de requisição para o processo P_i .
 - Se $\text{Requisição}_i[j] = k$, então o processo P_i deseja k instâncias do recurso R_j .
 - Quando uma requisição de recursos for feita para o processo P_i , as seguintes ações são realizadas:

Manipulação de Deadlock



- Algoritmo do banqueiro
 - Algoritmo da requisição de recursos
 1. Se $\text{Requisição}_i > \text{Necessário}_i$, levante uma condição de erro, pois o processo ultrapassou sua pretensão máxima, e aborte.
 2. Se $\text{Requisição}_i > \text{Disponível}$, P_i precisa esperar, pois os recursos não estão disponíveis.
 3. Faça o sistema fingir ter alocado os recursos requisitados ao processo P_i , modificando o estado da seguinte maneira:

Manipulação de Deadlock



- Algoritmo do banqueiro
 - Algoritmo da requisição de recursos
 - $Disponível = Disponível - Requisição_i$;
 - $Alocação_i = Alocação_i + Requisição_i$;
 - $Necessário_i = Necessário_i - Requisição_i$;
 - Verifique se o estado da alocação de recursos resultante é seguro ou inseguro utilizando o algoritmo do próximo slide.
 - Se seguro, a transação será completada e o processo P_i receberá a alocação de seus recursos.
 - Se inseguro, P_i terá de esperar por $Requisição_i$ e o antigo estado de alocação de recursos será restaurado.

Manipulação de Deadlock



- Algoritmo do banqueiro: estado seguro?
 - n processos, m recursos
 - 1. Sejam $Trabalho$ e Fim vetores de tamanho m e n , respectivamente. Inicialize $Trabalho = Disponível$ e $Fim[i] = false$ para todo i
 - 2. a) Encontre um i tal que
 - i. $Fim[i] == false$
 - ii. $Necessario[i] \leq Trabalho$se não houver tal i , vá para a etapa 4.
 - 3. a) $Trabalho = Trabalho + Alocação[i]$
b) $Fim[i] = true$
c) Vá para a etapa 2.
 - 4. Se $Fim[i] == true$ para todo i , então o sistema está em estado seguro.

Manipulação de Deadlock



- Atividade

- Faça um programa que implemente o algoritmo do banqueiro. Vários clientes (threads) requisitam recursos do banco. O banqueiro concederá uma requisição apenas se ela deixar o sistema em estado seguro. Uma requisição é negada se deixar o sistema em estado inseguro.