

Modulo I - Deadlocks

Programação Concorrente

Prof. Ismael H F Santos

Ementa

- **Programação Concorrente - Deadlocks**
 - The Deadlock Problem
 - Deadlock Characterization
 - Methods for Handling Deadlocks
 - Deadlock Prevention
 - Deadlock Avoidance - Banker's Algorithm
 - Deadlock Detection
 - Recovery from Deadlock

Objectives

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks
- To present a number of different methods for preventing or avoiding deadlocks in a computer system.

SCD – CO023

*Problema
De
Deadlock*



The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
 - System has 2 tape drives.
 - P_1 and P_2 each hold one tape drive and each needs another one.

The Deadlock Problem

- Example
 - semaphores A and B , initialized to 1

P_0	P_1
<code>wait (A);</code>	<code>wait(B)</code>
<code>wait (B);</code>	<code>wait(A)</code>

A Deadlock Example

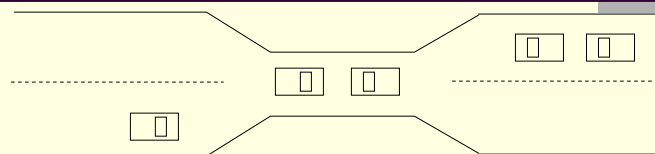
- In a given system, we have one printer and one plotter, and two processes A and B.

Process A	Process B
(1) -	(8) -
(2) Request (printer)	-
(3) -	(9) Request (plotter)
(4) Request (plotter)	(10) -
-	(11) Request (printer)
(5) Release (printer)	-
(6) Release (plotter)	(12) -
(7) -	-
-	(13) Release (printer)
-	(14) Release (plotter)
-	(15) -

Scenario 1: (1)-(7), (8)-(15) ==> No problem (no deadlock)

Scenario 2:
(1)-(3), interrupt, (8)-(11), B blocked, (4), A blocked ==> **Deadlock**

Bridge Crossing Example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

SCD – CO023

Caracterização
Deadlock



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

9

System Model

- Resource types R_1, R_2, \dots, R_m
CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
 - **request for a resource** is made; if the request is not granted, a process is blocked (and waits) until a resource is granted.
 - **usage of a resource for some time**; after resource being granted.
 - **release of resource**; when not any more needed.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

10

Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously.
 - **Mutual exclusion:** only one process at a time can use a resource.
 - **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
 - **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously (cont.)
 - **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_0 is waiting for a resource that is held by P_0 .
- Note above conditions are necessary but not sufficient for deadlock to occur !

Resource-Allocation Graph



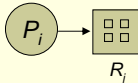
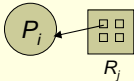
- Resource allocation graphs are useful in analyzing deadlock situations.
- A set of vertices V and a set of edges E
 - V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
 - request edge – directed edge $P_1 \rightarrow R_j$
 - assignment edge – directed edge $R_j \rightarrow P_i$

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

13

Resource-Allocation Graph (Cont.)

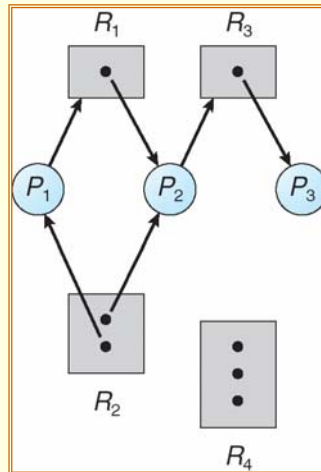
- Process 
- Resource Type with 4 instances 
- P_i requests instance of R_j 
- P_i is holding an instance of R_j 

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

14

Example of a Resource Allocation Graph



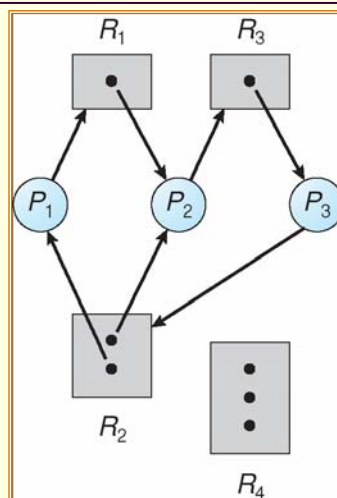
No cycle => no deadlock !

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

15

Resource Allocation Graph with a Deadlock



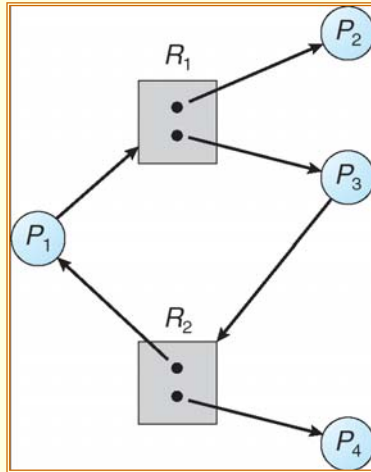
Deadlock exists !!!

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

16

Resource Allocation Graph with a Cycle But No Deadlock



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

17

Basic Facts

- If graph contains no cycles \Rightarrow no deadlock.
- If graph contains a cycle \Rightarrow
 - if only one instance per resource type, then deadlock.
 - if several instances per resource type, possibility of deadlock.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

18

SCD – CO023

Tratamento De Deadlock



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

19

Methods for Handling Deadlocks

- **Deadlock prevention:** Ensure that the system will *never* enter a deadlock state by making that at least one of the necessary deadlock conditions does not hold.
- **Deadlock avoidance:** Ensure that the system will *never* enter a deadlock state since O.S. never granting an unsafe request for resources; O.S. has to have some advanced information how resources are to be requested.
- **Deadlock detection and recovery:** Allow the system to enter a deadlock state and then recover.
- **Ignore the problem** and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

20

Deadlock Prevention

- **Restrain the ways request can be made**
 - **Mutual Exclusion** – not required for sharable resources; must hold for nonsharable resources.
 - **Solutions for some types of non-sharable resource:**
 - spool if you can, e.g. printer or plotter,
 - windows introduced to share one monitor.
 - **Unix does all of those, thus it takes some care of deadlock.**

Deadlock Prevention

- **Restrain the ways request can be made (cont.)**
 - **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources.
 - **Require process to request and be allocated all its resources before it begins execution, or**
 - **Allow process to request resources only when the process has none.**
 - **Because of low resource utilization or/and starvation, none of those approaches promising.**

Deadlock Prevention (Cont.)

- **No Preemption** – usually not promising, although applicable to resources whose state can be easily saved and restored later, e.g. CPU registers and memory space.
 - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
 - Preempted resources are added to the list of resources for which the process is waiting.
 - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Deadlock Prevention (Cont.)

- **Circular Wait** – an interesting and useful algorithm exists, that can be also applied in some different situations.
 - The algorithm imposes a total ordering of all resource types, i.e. each resource type has its unique number;
 - Processes can require resources any time but any request may be asking only for a resource numbered higher than any of currently held resources.
 - If this rule is followed by everybody, a circular wait is not possible, thus a deadlock is prevented. Note that the operating system can detect a process that doesn't follow the rule.

Deadlock Avoidance

- Deadlock avoidance algorithms normally requires that the system has some additional *a priori* information about process behavior. The simplest and most useful model requires that each process declares in advance the *maximum number* of resources of each type that it may need during its execution.
 - When a process requests an available resource, system must decide if immediate allocation leaves the system in a *safe state* or moves it into *unsafe state*.
 - The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
 - Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

25

Safe State

- System is in safe state if there exists a *safe sequence* of all processes.
- Sequence $\langle P_1, P_2, \dots, P_n \rangle$ is *safe* if for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j , with $j < i$.
 - If P_i resource needs are not immediately available, then P_i can wait until all P_j ($j < i$) have finished.
 - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
 - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

26

Basic Facts

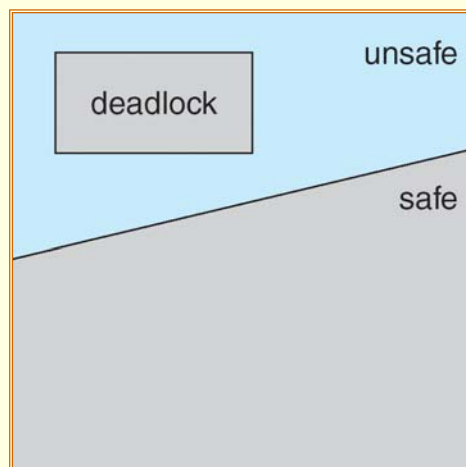
- If a system is in safe state \Rightarrow no deadlocks.
- If a system is in unsafe state \Rightarrow possibility of deadlock.
- **Deadlock Avoidance** \Rightarrow ensure that a system will never enter an unsafe state.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

27

Safe, Unsafe, Deadlock State



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

28

Resource-Allocation Graph Algorithm

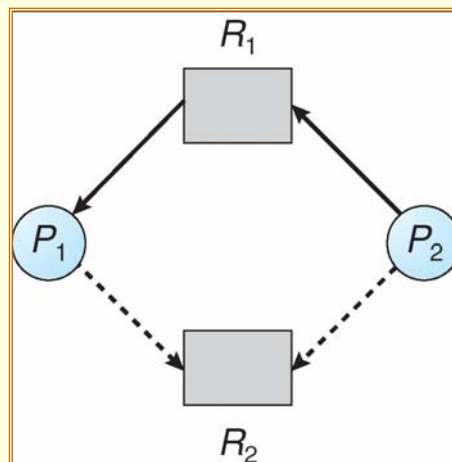
- Claim edge $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j ; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed *a priori* in the system.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

29

Resource-Allocation Graph For Deadlock Avoidance

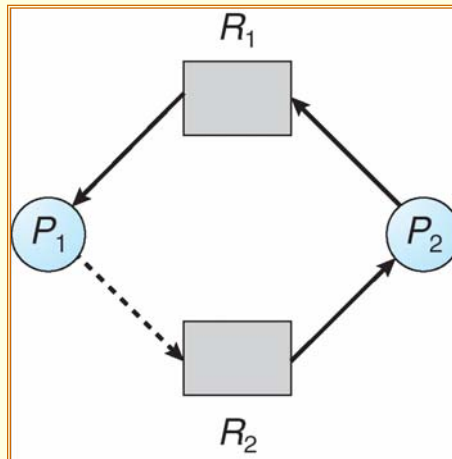


April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

30

Unsafe State In Resource-Allocation Graph



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

31

Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

32

Data Structures for the Banker's Algorithm

- Let n = number of processes, and m = number of resources types
 - *Available*: Vector of length m . If $Available[j] = k$, there are k instances of resource type R_j available.
 - *Max*: $n \times m$ matrix. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j .

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

33

Data Structures for the Banker's Algorithm

- *Allocation*: $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j .
- *Need*: $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$$Need[i,j] = Max[i,j] - Allocation[i,j].$$

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

34

Data Structures for the Banker's Algorithm (cont.)

- Definitions for a system with n processors and m resources:

$$\begin{array}{l} \text{Max - Avail matrix } A = (a_1 \ a_2 \ \dots \ a_m) \\ \text{Max - Claim matrix } B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} \\ \text{Current Allocation matrix } C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nm} \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{pmatrix} \end{array}$$

$$\begin{array}{l} \text{Current Avail matrix } D = (d_1 \ d_2 \ \dots \ d_m) = A - \sum_{i=1}^n C_i \\ \text{Current Need matrix } E = \begin{pmatrix} e_{11} & e_{12} & \dots & e_{1m} \\ e_{21} & e_{22} & \dots & e_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1} & e_{n2} & \dots & e_{nm} \end{pmatrix} = B - C = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix} \\ \text{Request vector } F_i = (f_{i1} \ f_{i2} \ \dots \ f_{im}) \end{array}$$

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

35

Safety Algorithm

1. Let *Work* and *Finish* be vectors of length m and n , respectively. Initialize:

Work = Available

Finish [i] = false for $i = 1, 3, \dots, n$.

2. Find and i such that both:

(a) *Finish* [i] = false

(b) $Need_i \leq Work$

If no such i exists, go to step 4.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

36

Safety Algorithm

3. $Work = Work + Allocation;$
 $Finish[i] = true$
go to step 2.

4. If $Finish [i] == true$ for all i , then the system is in a safe state.

Safety Algorithm -pseudocode

■ Algoritmo Safety

$Work[i] = Available[i]$ ($i=1,m$)

Enquanto $\exists i$ tq $Finish[i] = False \ \&\& \ Need[i,j] \leq Work[j]$ ($j=1,m$)

$Work[k] += Allocated[i,k];$ ($k=1,m$)

$Finish[i] = True;$

Fim_Enquanto

Se $\exists k$ tq $Finish[k] = False$ então

UNSAFE /* e os processos com $Finish[k] = false$

 Senão /* estão em Deadlock */

SAFE

Fim_Se

Resource-Request Algorithm for Process P_i

$Request$ = request vector for process P_i .
If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j .

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

39

Resource-Request Algorithm for process P_i

3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$Available = Available - Request_i$;

$Allocation_i = Allocation_i + Request_i$;

$Need_i = Need_i - Request_i$;

4. Execute Safety Algorithm

- If safe \Rightarrow the resources are allocated to P_i .
- If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored, ie, roll back the updates of step 3.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

40

Banker's Algorithm -pseudocode

- **Banker's Algorithm**
Request(P_i) = Request[i,k] -> Processo P_i pede Request[i,k] instancias do recurso R_k
Se Request[i,k] > Need[l,k] ($k=1,m$) Então
ERRO;
Senão
Se Request[i,k] > Available[k] ($k=1,m$) Então
 P_i entra em Wait
Senão
 // Simula entrega de recursos para P_i
 Available[k] -= Request[i,k] ($k=1,m$)
 Allocated[i,k] += Request[i,k] ($k=1,m$)
 Need[i,k] -= Request[i,k] ($k=1,m$)
 estado = Safety_Algorithm();
 Se estado == SAFE Então
 P_i obtem permissao para alocar os recursos
 Senão
 P_i entra em wait e faz-se o rollback em Available, Allocated e Need
 Fim_Se
Fim_Se
Fim_Se

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

41

Example of Banker's Algorithm

- 5 processes P_0 through P_4 ; 3 resource types A (10 instances), B (5 instances, and C (7 instances).

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

42

Example of Banker's Algorithm

- Snapshot at time T_0 :

	<u>Allocation</u>			<u>Max</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
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			

Example (Cont.)

- The content of the matrix. Need is defined to be 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

Example (Cont.)

- The system is in a safe state since the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies safety criteria.

Example P_1 Request (1,0,2) (Cont.)

- Check that Request \leq Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true.

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
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	

Example P_1 Request (1,0,2) (Cont.)

- Executing safety algorithm shows that sequence $\langle P1, P3, P4, P0, P2 \rangle$ satisfies safety requirement.
- Can request for (3,3,0) by P4 be granted?
- Can request for (0,2,0) by P0 be granted?

Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Single Instance of Each Resource Type

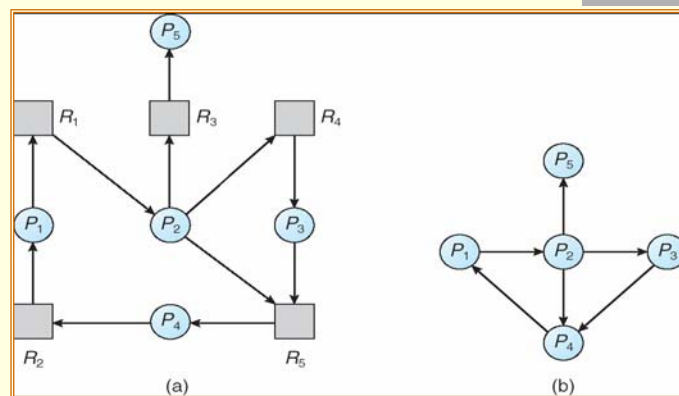
- Maintain *wait-for* graph
 - Nodes are processes.
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j .
- Periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

49

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

50

Several Instances of a Resource Type

- *Available*: A vector of length m indicates the number of available resources of each type.
- *Allocation*: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- *Request*: An $n \times m$ matrix indicates the current request of each process. If $Request[i] = k$, then process P_i is requesting k more instances of resource type R_j .

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

51

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length m and n , respectively Initialize:
 - (a) $Work = Available$
 - (b) For $i = 1, 2, \dots, n$, if $Allocation_i \neq 0$, then $Finish[i] = false$; otherwise, $Finish[i] = true$.
2. Find an index i such that both:
 - (a) $Finish[i] == false$
 - (b) $Request_i \leq Work$

If no such i exists, go to step 4.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

52

Detection Algorithm (Cont.)

3. $Work = Work + Allocation_i$
 $Finish[i] = true$
go to step 2.

4. If $Finish[i] == false$, for some i , $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked.

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances).
- Snapshot at time T_0 :

	<u>Allocation</u>			<u>Request</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0

Example of Detection Algorithm

AllocationRequestAvailable

	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

- Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = \text{true}$ for all i .

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

55

Example (Cont.)

- P_2 requests an additional instance of type C.

Request

	A	B	C
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

56

Example (Cont.)

- State of system?
 - Can reclaim resources held by process P_0 , but insufficient resources to fulfill other processes; requests.
 - Deadlock exists, consisting of processes P_1 , P_2 , P_3 , and P_4 .

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
 - How often a deadlock is likely to occur?
 - How many processes will need to be rolled back?
 - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

SCD – CO023

Recuperação
De
Deadlock



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

59

Recovery from Deadlock: Process Termination

- **Process Termination:**
 - Abort all deadlocked processes.
 - Abort one process at a time until the deadlock cycle is eliminated. In which order should we choose to abort?
 - Priority of the process.
 - How long process has computed, and how much longer to completion.
 - Resources the process has used.
 - Resources process needs to complete.
 - How many processes will need to be terminated.
 - Is process interactive or batch?
- **Rollback – return to some safe state, and restart processes for that state.**

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

60

Recovery from Deadlock: Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

61

Combined Approach to Deadlock Handling

- Combine the three basic approaches
 - prevention
 - avoidance
 - detectionallowing the use of the optimal approach for each of resources in the system.
- Partition resources into hierarchically ordered classes.
- Use most appropriate technique for handling deadlocks within each class.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

62

Deadlock Detection and Recovery

- Allow a system to enter deadlock state.
- **Deadlock detection algorithm** needed to detect deadlock, and then some **recovery scheme** has to apply.
- When, and how often, to invoke a deadlock detection algorithm?
- If a deadlock detection algorithm is not invoked on time, there may be many cycles in the resource-allocation graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

63

Deadlock Problem – Exercice

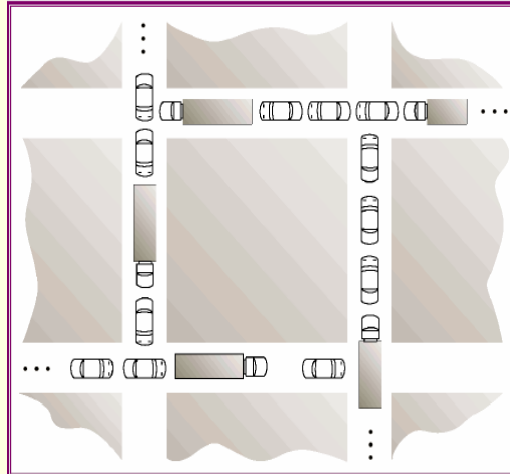
- In an electronic fund transfer system, there are hundreds of identical processes that work as follows. Each process reads an input line specifying the account to be credited, the account to be debited and an amount of money. The process first locks both accounts and transfers the money, releasing the locks when done.
- When a process attempts to lock an account already locked by another process, it will be blocked and it will stay blocked until the account is released.
- With many processes running in parallel, there is a very real danger that having locked the account x, a process will be unable to lock the account y (thus it will be blocked) because y has been locked by another process now waiting (and it is blocked) for x. **Deadlock!!!**
- Devise a schema that is deadlock free.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

64

Traffic Deadlock – Exercise



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

65