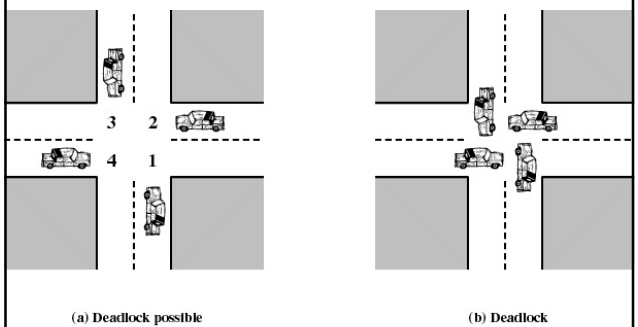
Concurrency: Deadlock and Starvation

Chapter 6, Livro do William Stallings

Sistemas de Operação, 2004-2005

Deadlock

- Permanent blocking of a set of processes that either compete for system resources or communicate with each other.
- No efficient solution.
- Involve conflicting needs for resources by two or more processes.



(a) Deadlock possible (b) Deadlock

Figure 6.1 Illustration of Deadlock

Example

Process P	Process Q
Get A	Get B
Get B	Get A
....	....
Release A	Release B
Release B	Release A

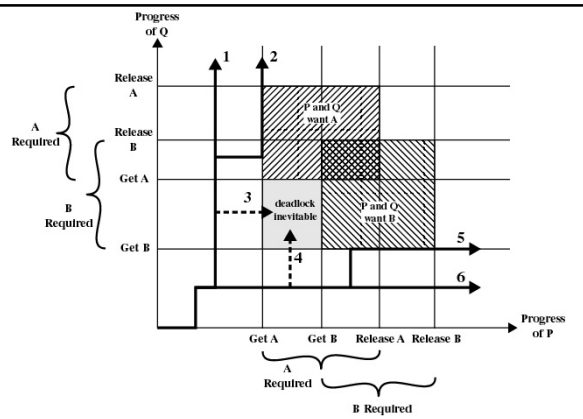


Figure 6.2 Example of Deadlock [BACO98]

Example 2

Process P	Process Q
Get A ... Release A .... Get B .... Release B	Get B Get A .... Release B Release A

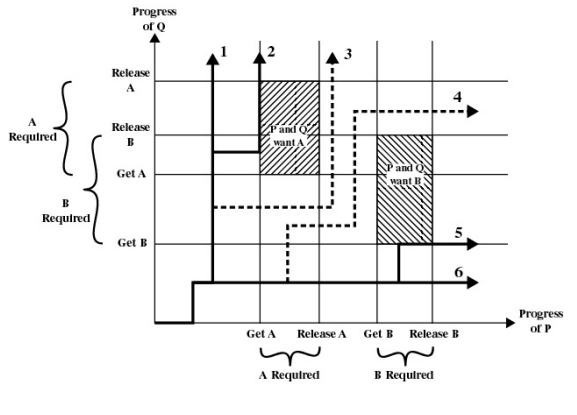


Figure 6.3 Example of No Deadlock [BACO98]

Reusable Resources

- Used by one process at a time and not depleted by that use.
- Processes obtain resources that they later release for reuse by other processes.
- Processors, I/O channels, main and secondary memory, files, databases, and semaphores.
- Deadlock occurs if each process holds one resource and requests the other.

Consumable Resources

- Created (produced) and destroyed (consumed) by a process.
- Interrupts, signals, messages, and information in I/O buffers.
- Deadlock may occur if a Receive message is blocking.
- May take a rare combination of events to cause deadlock.

Another Example of Deadlock

- Deadlock occurs if receive is blocking

```
Process P1          Process P2
....                ....
Receive(P2)         Receive(P1)
...                 ...
Send(P2,M1)         Send(P1,M2)
```

4 Conditions for Deadlock

- **Mutual exclusion**
  - only one process may use a resource at a time
- **Hold-and-wait**
  - A process request all of its required resources at one time

4 Conditions for Deadlock

- **No-preemption**
  - If a process holding certain resources is denied a further request, that process must release its original resources.
  - If a process requests a resource that is currently held by another process, the operating system may preempt the second process and require it to release its resources.

4 Conditions for Deadlock

- **Circular Wait**
  - Prevented by defining a linear ordering of resource types

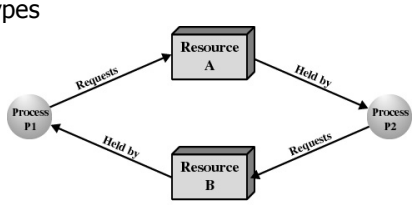
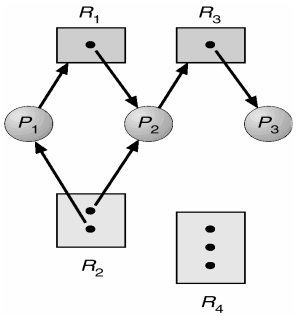
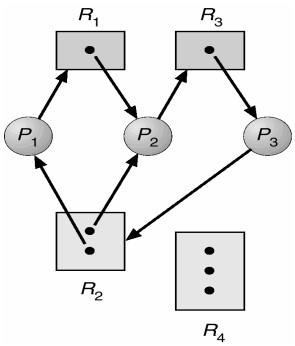


Figure 6.5 Circular Wait

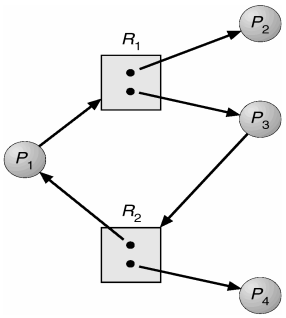
Resource Graph (not deadlock)



Resource Graph (deadlock)



Resource Graph (with a cycle but no deadlock)



Graph Analysis

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

How to Prevent Circular Wait?

- Define a linear ordering of resources:
  R1, R2, R3,...., Rn
- If (i < j):

Acquire (Ri) Acquire (Rj)

OK

Acquire (Rj) Acquire (Ri)

NOT OK

Examples

Processo P1:	**Processo P2:**	**Processo P3:**	**Processo P4:**
Sem_wait(A);	Sem_wait(B);	Sem_wait(C);	Sem_wait(B);
Sem_wait(C);	Sem_wait(A);	Sem_wait(A);	Sem_wait(D);
.....	.....	.....	.....
Sem_signal(C);	Sem_signal(A);	Sem_signal(A);	Sem_signal(D);
Sem_signal(A);	Sem_signal(B);	Sem_signal(C);	Sem_signal(B);

Processo P1:	**Processo P2:**	**Processo P3:**	**Processo P4:**
Sem_wait(A);	Sem_wait(B);	Sem_wait(C);	.....
Sem_wait(C);	Sem_wait(A);	Sem_wait(A);	Sem_wait(D);
.....	.....	.....	.....
Sem_signal(C);	Sem_signal(A);	Sem_signal(A);	Sem_signal(D);
Sem_signal(A);	Sem_signal(B);	Sem_signal(C);	Sem_signal(C);

NO DEADLOCK

Processo P1:	**Processo P2:**	**Processo P3:**	**Processo P4:**
Sem_wait(A);	Sem_wait(A);	Sem_wait(A);	Sem_wait(B);
Sem_wait(C);	Sem_wait(B);	Sem_wait(C);	Sem_wait(D);
.....	.....	.....	.....
Sem_signal(C);	Sem_signal(B);	Sem_signal(C);	Sem_signal(D);
Sem_signal(A);	Sem_signal(A);	Sem_signal(A);	Sem_signal(B);

Processo P1:	**Processo P2:**	**Processo P3:**	**Processo P4:**
Sem_wait(A);	Sem_wait(B);	Sem_wait(C);	.....
Sem_wait(C);	Sem_wait(A);	Sem_wait(A);	Sem_wait(D);
.....	.....	.....	.....
Sem_signal(C);	Sem_signal(A);	Sem_signal(A);	Sem_signal(D);
Sem_signal(A);	Sem_signal(B);	Sem_signal(C);	Sem_signal(C);

Deadlock Avoidance

Two Approaches to Deadlock Avoidance

- Do not start a process if its demands might lead to deadlock.
- Do not grant an incremental resource request to a process if this allocation might lead to deadlock.

Resource Allocation Denial

- Referred to as the Banker's Algorithm
- State of the system is the current allocation of resources to process.
- Safe state is where there is at least one sequence that does not result in deadlock.
- Unsafe state is a state that is not safe.

Determination of a Safe State

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim Matrix

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation Matrix

R1	R2	R3
9	3	6

Resource Vector

R1	R2	R3
0	1	1

Available Vector

(a) Initial state

Question: can any of the 4 processes run into completion with the available resources?

What about P1?

What about P2?

Determination of a Safe State

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim Matrix

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation Matrix

R1	R2	R3
6	2	3

Available Vector

(b) P2 runs to completion

P2 Runs to Completion

Determination of a Safe State

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim Matrix

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation Matrix

R1	R2	R3
7	2	3

Available Vector

(c) P1 runs to completion

P1 Runs to Completion

Determination of a Safe State

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim Matrix

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation Matrix

R1	R2	R3
9	3	4

Available Vector

(d) P3 runs to completion

P3 Runs to Completion

Deadlock Avoidance Strategy

- When a process makes a request for a set of resources assume that the request is granted, update the system state accordingly and then determine if the result is a SAFE STATE.
- If so, grant the request.
- If not, block the process until it is safe to grant the request.

Determination of an Unsafe State

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim Matrix

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation Matrix

R1	R2	R3
9	3	6

Resource Vector

R1	R2	R3
1	1	2

Available Vector

(a) Initial state

Assume that P1 makes a request for one unit of R1 and one unit of R3.

If the request is granted see the next Figure...

Determination of an Unsafe State

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim Matrix

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation Matrix

R1	R2	R3
0	1	1

Available Vector

(b) P1 requests one unit each of R1 and R3

Is this a SAFE STATE?

No, because each process will need at least an additional unit of R1 and there are none available.

Thereby, to avoid a deadlock the request of P1 should be denied.

Unsafe State != Deadlocked State

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim Matrix

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation Matrix

R1	R2	R3
0	1	1

Available Vector

(b) P1 requests one unit each of R1 and R3

This is not a DEADLOCKED STATE!

It only has the potential for Deadlock.

Deadlock Detection

Algorithm

1. Mark each process that has a row in the Allocation Matriz of all zeros.
2. Initialize a temporary vector W to equal the Available Vector.
3. Find an index i such that process i is currently unmarked and the i^{th} row of Q is less than or equal to W. That is, $Q_{ik} <= W_k$ (k=1...m).
4. If no such row is found terminate the algorithm.
5. If such a row is found mark process i and add the corresponding row of the allocation matrix to W. That is: set $W_k = W_k + A_{ik}$ (k=1...m). Return to step 3.

Q = Request Matrix; A = Allocation Matrix.

Deadlock Detection

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request Matrix Q

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation Matrix A

R1	R2	R3	R4	R5
2	1	1	2	1

Resource Vector

R1	R2	R3	R4	R5
0	0	0	0	1

Available Vector

Figure 6.9 Example for Deadlock Detection

Illustrate the Deadlock Detection Algorithm

- Mark P4, because P4 has no allocated resources.
- Set W = [0 0 0 0 1]
- The request of process P3 is less than or equal to W, so mark P3 and set:
  W = W + [0 0 0 1 0] = [0 0 0 1 1]
- No other unmarked process has a row in Q that is less than or equal to W. Therefore, terminate the algorithm.

P1 and P2 are unmarked → these processes are deadlocked!

Recovery from Deadlock…

- Abort all deadlocked processes.
- Back up each deadlocked process to some previously defined checkpoint, and restart all process
  - original deadlock may occur
- Successively abort deadlocked processes until deadlock no longer exists.
- Successively preempt resources until deadlock no longer exists.

QUIZ

Considere o seguinte cenário onde existem 4 processos a usar 4 recursos de sistema. A matriz de pedidos (*Request*), a matriz de recursos atribuídos (*Allocation*) e o vector de recursos disponíveis (*Available*) estão representados na Figura seguinte:

Matriz Allocation A:	**0 0 1 0**
	0 0 1 0
	2 0 0 1
	0 1 2 0
Matriz de Requests Q:	**1 0 2 0**
	2 0 0 1
	1 0 1 1
	2 1 0 0
Vector Available:	**2 1 0 0**

Aplique o algoritmo de detecção de deadlocks e indique se existe ou não algum deadlock entre os processos. (Não dê uma resposta binária: explique a aplicação do algoritmo).