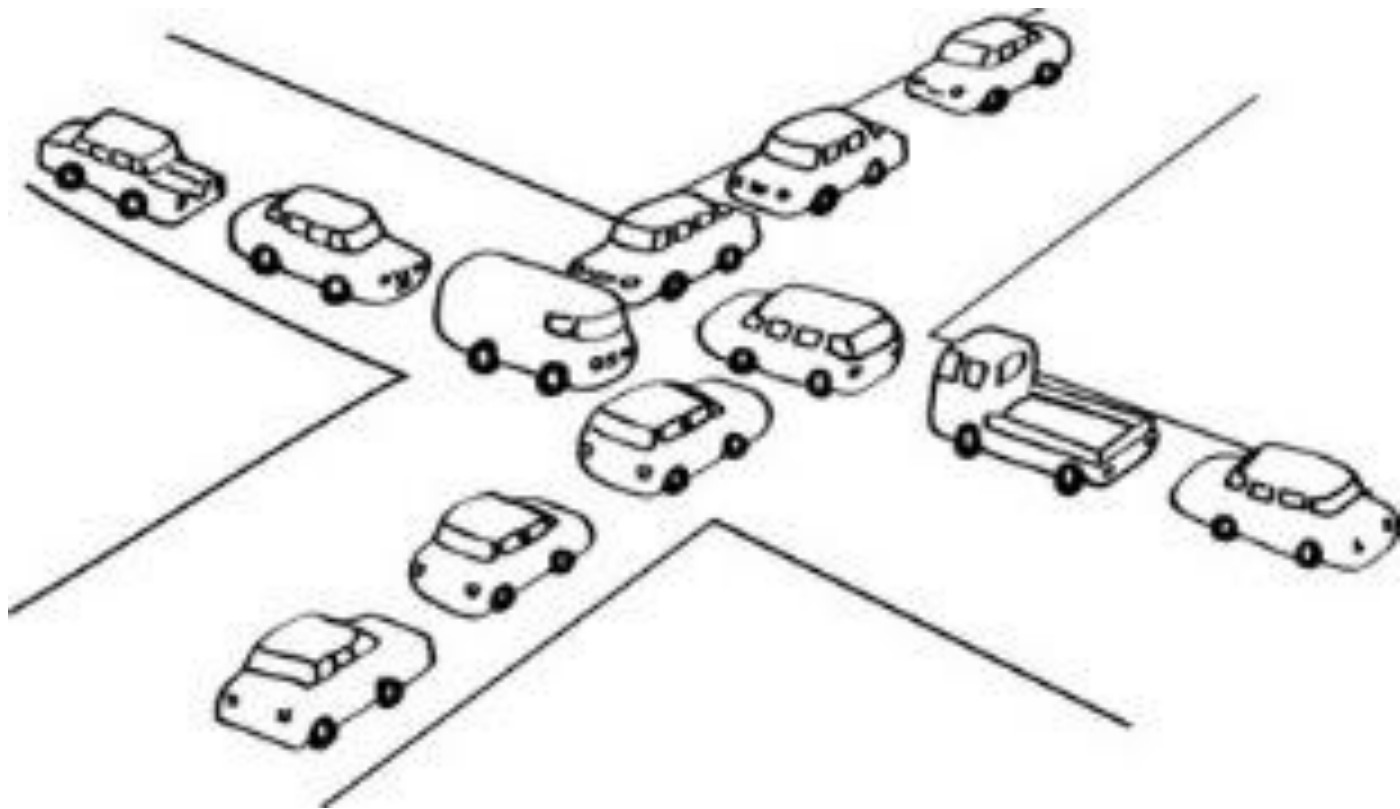


Deadlocks



A process may utilize a resource
in only the following sequence:

System Model

1. Request
2. Use
3. Release

Necessary condition ของ **event A** หมายถึง

ถ้าเกิด **event A** จะต้องพบ **necessary condition** ด้วย

แต่พบ **necessary condition** อาจจะไม่เกิด **event A** ก็ได้

เช่น ถ้านกบินได้ นกต้องมีปีก แต่มีปีกอาจจะบินไม่ได้ (เพนกวิน)

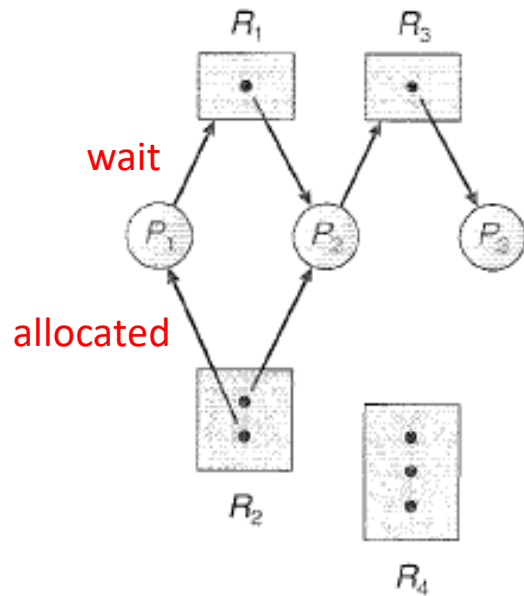
ไฟมี **necessary condition** คือ ความร้อน เชื้อเพลิง และอากาศ

ขาด **condition** ใด **condition** หนึ่งไป ไฟก็จะดับ เป็นหลักการดับเพลิง

Necessary Conditions

1. Mutual exclusion (non-sharable)
2. Hold and wait (holding at least one resource, and wait)
3. No preemption (resources cannot be preempted)
4. Circular wait

Resource-Allocation Graph



- The sets P , R , and E :
 - $P = \{P_1, P_2, P_3\}$
 - $R = \{R_1, R_2, R_3, R_4\}$
 - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$
- Resource instances:
 - One instance of resource type R_1
 - Two instances of resource type R_2
 - One instance of resource type R_3
 - Three instances of resource type R_4

Figure 7.2 Resource-allocation graph.

Resource-Allocation Graph

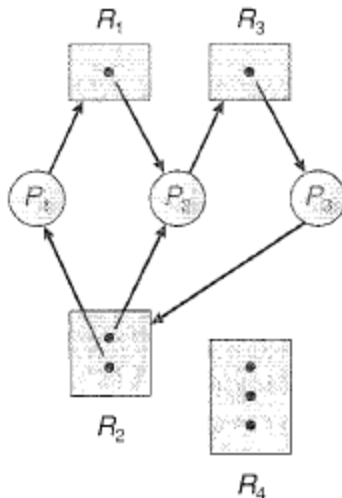


Figure 7.3 Resource-allocation graph with a deadlock.

Circular Wait

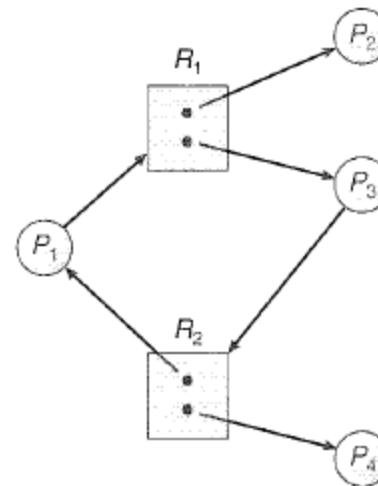


Figure 7.4 Resource-allocation graph with a cycle but no deadlock.

Necessary but not sufficient!

Methods for Handling Deadlocks

1. Deadlock prevention

Prevent at least one of the necessary conditions.

2. Deadlock avoidance จะไม่พยายามไปแก้ necessary condition

Information in advance (wait/no wait requests).

3. Deadlock detection and recovery

Deadlocks arise, detected, and recovered.

4. Do nothing

Performance deterioration, stop functioning, and need to be manually restarted.

Deadlock Prevention

1. Mutual exclusion
Read-only file (sharable)
CPU, memory (non-sharable)
Intrinsically non-sharable.
2. Hold and wait
Whenever a process requests a resource, it does not hold any other resources.
Request resources only when having none.
 - 2.1 Request all resources at the beginning (low utilization)
 - 2.2 Release all resources before making a request (cannot always release)A process that needs several popular resources may have to wait indefinitely (starvation).
Often applied to resources whose state can be easily saved and restored later.
เช่น CPU และ memory
3. No preemption
If a request fails, the resources will be preempted.
4. Circular wait
Request resources in increasing order.

$F(\text{tape drive}) = 1$

$F(\text{disk drive}) = 5$

$F(\text{printer}) = 12$

Each process can request resources only in an increasing order.

Rule 1: Process can request R_j that $F(R_j) > F(R_i)$

Rule 2: If requesting R_j , process must have released any R_i that $F(R_i) \geq F(R_j)$.

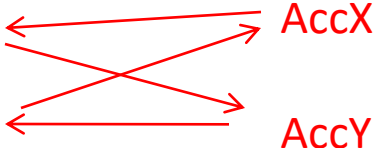
It must also be noted that if several instances of the same resource type are needed, a single request for all of them must be issued.

If the two rules are used, then the circular-wait condition cannot hold (proof by contradiction).

Wrong Implementation

```
void transaction(Account from, Account to, double amount) {  
  
    Semaphore lock1, lock2;  
    lock1 = getLock(from);  
    lock2 = getLock(to);  
  
    wait(lock1);    ล็อคตามลำดับแล้ว  
    wait(lock2);    ล็อค from ก่อน แล้วค่อยล็อค to  
  
    withdraw(from, amount);  
    deposit(to, amount);  
  
    signal(lock1);  
    signal(lock2);  
}
```

Transaction(AccX, AccY, 25);
Transaction(AccY, AccX, 50);



The diagram illustrates a deadlock state. On the left, two transaction calls are listed: Transaction(AccX, AccY, 25); and Transaction(AccY, AccX, 50);. On the right, two account names are listed: AccX and AccY. Red arrows indicate the locking requirements: the first transaction needs locks on both AccX and AccY, and the second transaction also needs locks on both AccX and AccY. The arrows cross, showing that each transaction is waiting for the other to finish, which is why they can never complete.

How to correct the program to prevent deadlock?

Deadlock Avoidance

A state is *safe* if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence. A sequence of processes $\langle P_1, P_2, \dots, P_n \rangle$ is a safe sequence for the current allocation state if, for each P_i , the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j , with $j < i$. In this situation, if the resources that P_i needs are not immediately available, then P_i can wait until all P_j have finished. When they have finished, P_i can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate. When P_i terminates, P_{i+1} can obtain its needed resources, and so on. If no such sequence exists, then the system state is said to be *unsafe*.

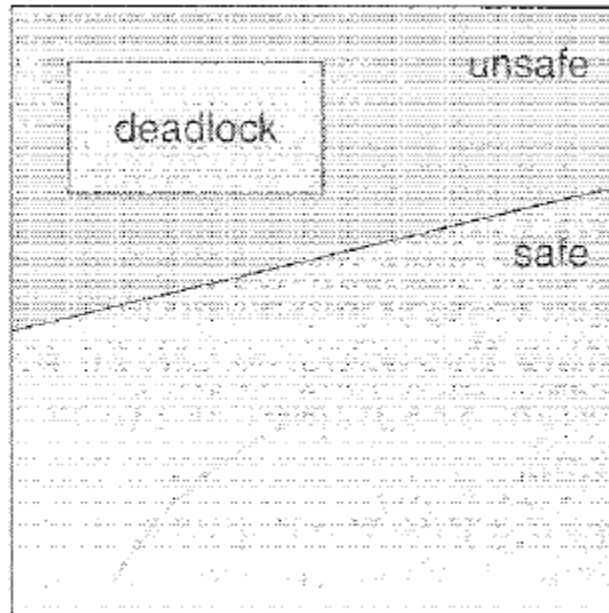


Figure 7.5 Safe, unsafe, and deadlock state spaces.

	<u>Maximum Needs</u>	<u>Current Needs</u>	
P_0	10	5	
P_1	4	2	เสียง request ที่
P_2	9	2	ให้ไปแล้ว ไม่ safe
			→ 3

Total 12 tape drives.

Safe sequence: $\langle P_1, P_0, P_2 \rangle$

มี tape drive ว่างอยู่ 3 ตัว

terminate P_1 แล้วมี tape drive ว่าง 5 ตัว

terminate P_0 แล้วมี tape drive ว่าง 10 ตัว

terminate P_2 แล้วมี tape drive ว่าง 12 ตัว

Suppose that P_2 requests 1 more tape drive and is allocated.
System is no longer in safe state.

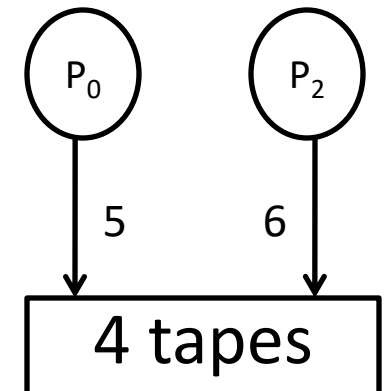
Deadlock may occur, for instance,

P_1 is allocated and returns all tape drives
($12 - 5 - 0 - 3$) = 4 tapes are available).

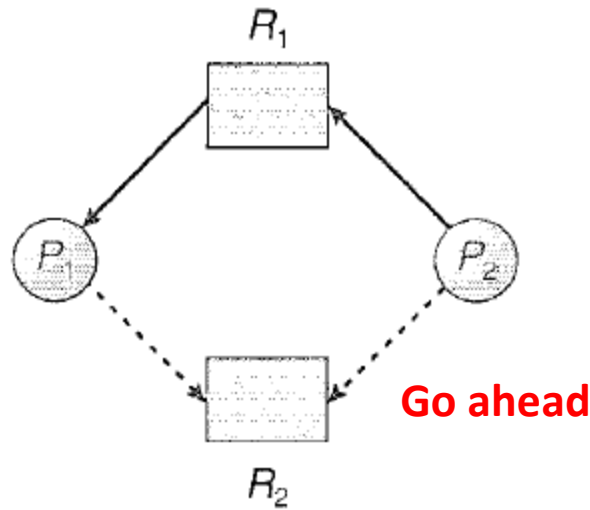
P_0 needs $10 - 5 = 5$ tape drives to terminate.

P_2 needs $9 - 3 = 6$ tape drives to terminate.

Deadlock!



Resource-Allocation-Graph Algorithm



Cycle-detection algorithm
 $O(n^2)$ where n is number of processes.

Figure 7.6 Resource-allocation graph for deadlock avoidance.

วิธีนี้ใช้กับ resource ที่มี
multiple instance ไม่ได้ !!!

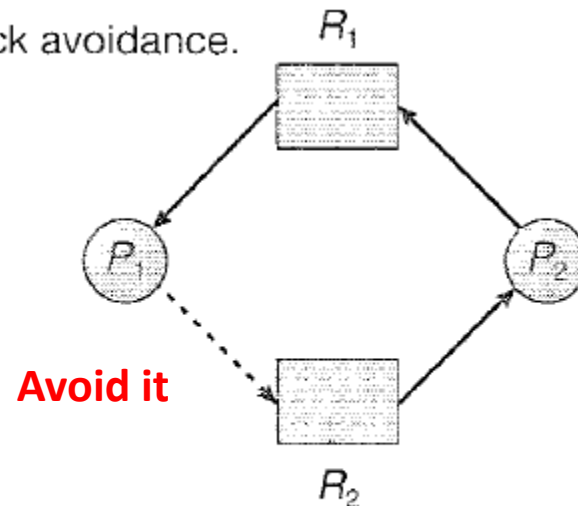


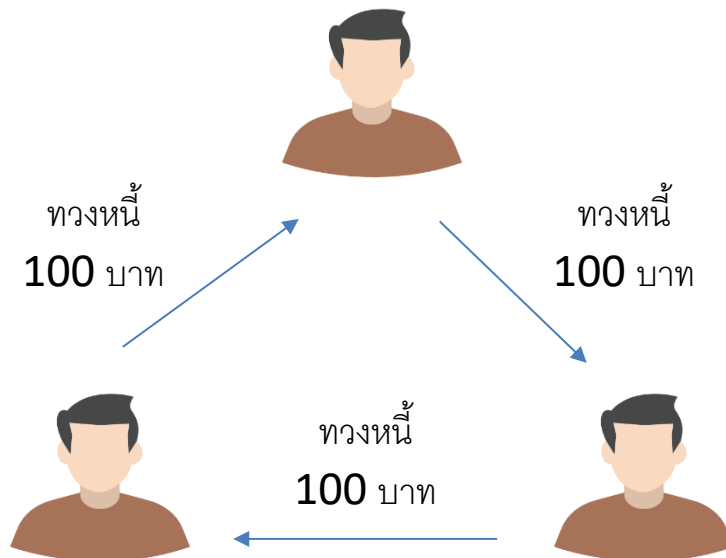
Figure 7.7 An unsafe state in a resource-allocation graph.

Banker's Algorithm

ทำ deadlock avoidance

โดยเช็คว่าย้ายอมให้ request จะอยู่ในสถานะ safe หรือไม่

n	number of processes
m	number of resource types
Available	vector of length m (number of available instances)
Max	n x m matrix (maximum demand)
Allocation	n x m matrix (number of allocated instances)
Need	n x m matrix (remaining resource need)



ทุกคนมีเงินไม่ถึง 100 บาท

Safety? Algorithm

1. Avail = (3, 3, ..., 2) // vector of length m
Term = (false, false, ..., false) // vector of length n

true คือ process ทำงานเสร็จ, false คือยัง

2. Find an index i such that both
 - a. Term[i] = false
 - b. Need_i ≤ Avail

Main idea:
ไล่หา **safe sequence**

3. Avail = Avail + Alloc_i
Term[i] = true;
Goto step 2

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

Time complexity = $O(mn^2)$ สูงกว่า **cycle detection algorithm**

Resource-Request Algorithm

Main idea:

เมื่อมี **request** ก็ลองสมมติว่าให้ดูก่อน
แล้วรัน **safety algo** ว่า **safe** หรือไม่

	<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	P_0	7 4 3
P_1	2 0 0	3 2 2		P_1	1 2 2
P_2	3 0 2	9 0 2		P_2	6 0 0
P_3	2 1 1	2 2 2		P_3	0 1 1
P_4	0 0 2	4 3 3		P_4	4 3 1

1

Safe (มี safe seq ≥ 1)
 $\langle P_1, P_3, P_4, P_2, P_0 \rangle$

2 P_1 requests (1, 0, 2) and granted.

	<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 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

3

Safe, then granted
 $\langle P_1, P_3, P_4, P_0, P_2 \rangle$

4 P_4 requests (3, 3, 0) not enough available resources
 P_0 requests (0, 2, 0) unsafe

Deadlock Detection

- Single instance
Resource-allocation graph called wait-for graph.

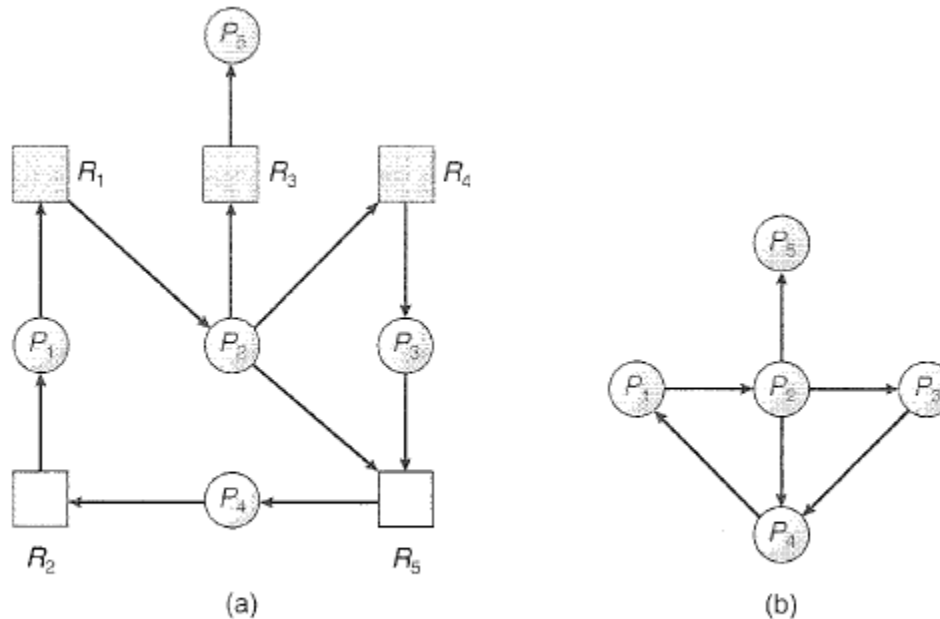


Figure 7.8 (a) Resource-allocation graph. (b) Corresponding wait-for graph.

Cycle = Deadlock
เฉพาะกรณี single instance

- Several instance
Deadlock-detection algorithm (similar to Safety algo).

Deadlock-Detection Algorithm

1. Avail = (3, 3, ..., 2) // vector of length m
Term = (false, false, ..., false) // vector of length n

2. Find an index i such that both

a. Term[i] = false

b. Request $_i \leq$ Avail

3. Avail = Avail + Alloc $_i$

Term[i] = true;

Goto step 2

4. If Term[i] == false for some i , $0 \leq i < n$, then P_i is deadlocked.

Time complexity = $O(mn^2)$

Main idea:

สมมติกรณี **best case**

ว่าทุก **process** จะไม่ **request** เพิ่ม
พยายามไล่ให้ทุก **process** ทำงานเสร็จ
ถ้าทำไม่ได้ ก็เกิด **deadlock** แล้ว

ไม่ต้องดู max หรือ need

	<u>Allocation</u>	<u>Current Request</u>	<u>Available</u>
	<i>A B C</i>	<i>A B C</i>	<i>A B C</i>
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	

No deadlock

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>
	<i>A B C</i>	<i>A B C</i>	<i>A B C</i>
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 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Deadlock!

Detection-Algorithm Usage ต้องพิจารณา

1. How often is a deadlock likely to occur?
2. How many processes will be affected by deadlock when it happens?

Recovery from Deadlock

1. Process Termination

Abort all deadlocked processes (ถ้า deadlock ไม่บ่อย และกระทบ process จำนวนไม่มาก).

Abort one process at a time until the deadlock cycle is eliminated.

2. Resource Preemption

Selecting a victim.

Rollback (due to resource preempted, total rollback = restart).

Starvation (re-preempt from the same process over and over).

Which method handles deadlock from Linux, Unix, Windows 7, and 10?



Larye Parkins, Linux user & sysadmin since 1994

Answered Dec 10, 2018



Windows SQL server has a lock monitor that attempts to detect and resolve deadlocks automatically, only possible because databases have a rollback feature. Unix and Linux, because they deal with asynchronous independent processes, don't have any internal mechanism for dealing with deadlocks, the type of monitoring used by the more controlled database model is simply not cost effective for the rare times deadlocks occur. Deadlocks are more likely to occur between different branches of a threaded or forked process, (i.e., like a database) and are therefore preventable by careful design and require manual intervention to break when they do occur.

ใน database แก้ deadlock ได้ เพราะ rollback transaction ได้
โดย rollback transaction ทั้งหมด/หรือทีละหนึ่ง จนกว่า deadlock จะหายไป