政大資科系

作業系統

Operating System

廖峻鋒

cfliao@nccu.edu.tw

Operating System

Deadlock

Chun-Feng Liao

廖峻鋒

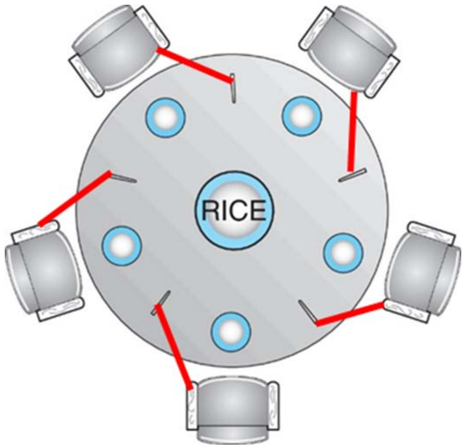
Department of Computer Science

National Chengchi University

Definition

- A set of threads is in a deadlocked state when
  - All thread in the set is waiting for an event that can be caused only by another thread in the set
  - Key events: resource acquisition and release

```
while (true) {
   wait(chopstick[i]);
   wait(chopstick[(i+1) % 5]);
     . . .
   /* eat for a while */
     . . .
   signal(chopstick[i]);
   signal(chopstick[(i+1) % 5]);
     . . .
   /* think for awhile */
     . . .
}
```



P.321

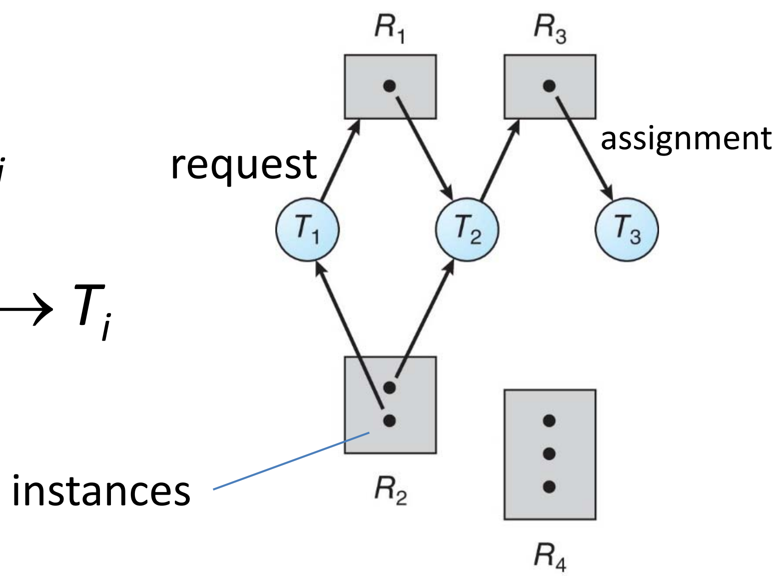
Necessary Conditions

- 四個條件都同時成立，才會有deadlock
  - 免除其中一個條件就可避免deadlock
- deadlock四要件
  - Mutual exclusion: only one at a time can use a resource
  - Hold & Wait: someone holding some resources and is waiting for another resource
  - No preemption: a resource can only be released voluntarily
  - Circular wait: $\exists\{T_0, T_1, \ldots, T_n\} \Rightarrow T_0 \rightarrow T_1 \rightarrow \ldots \rightarrow T_n \rightarrow T_0$

Resource-Allocation Graph

A set of vertices *V* and a set of edges *E*.

- V : two types
  - $T = \{T_1, T_2, \ldots, T_n\}$, the set consisting of all the threads in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $T_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow T_i$



If the graph contains a cycle, a deadlock **may exist!**

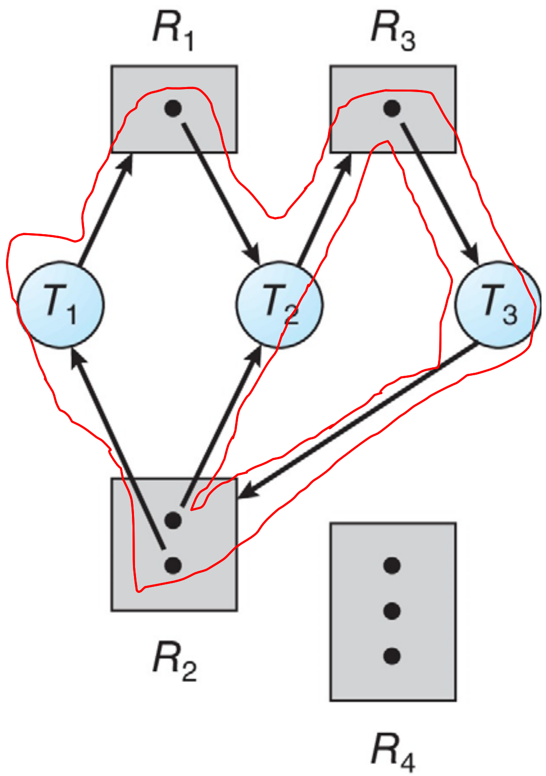
Basic Facts

- If graph contains no cycle → no deadlock
  - Circular wait cannot be held
- If graph contains a cycle:
  - if one instance per resource type → deadlock
  - if multiple instances per resource type → possibility of deadlock

Example

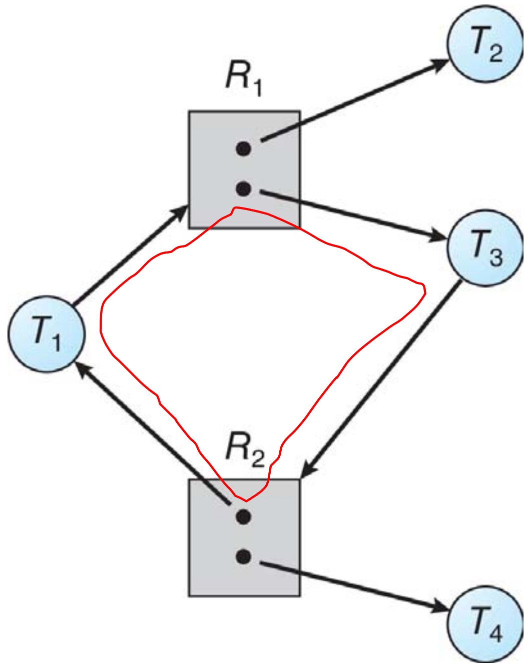
Deadlock: All threads in the set are waiting for events that can be caused only by another thread in the set

觀察重點: 先看有等資源的→評估是否可能等到



Deadlock

All threads are waiting for other threads



No deadlock

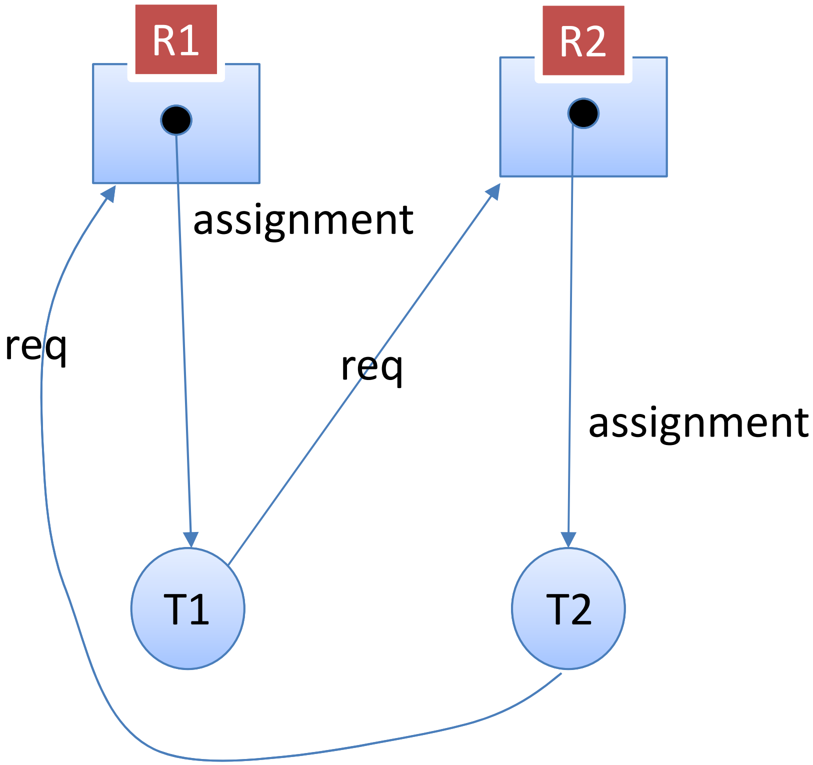
T2, T4 are not waiting for other threads

P.319

Example

```
pthread_mutex_t first_mutex;     R1
pthread_mutex_t second_mutex;    R2

pthread_mutex_init(&first_mutex,NULL);
pthread_mutex_init(&second_mutex,NULL);
```



T1

```
/* thread_one runs in this function */
void *do_work_one(void *param)
{
   pthread_mutex_lock(&first_mutex);
   pthread_mutex_lock(&second_mutex);
   /**
    * Do some work
    */
   pthread_mutex_unlock(&second_mutex);
   pthread_mutex_unlock(&first_mutex);

   pthread_exit(0);
}
```

T2

```
/* thread_two runs in this function */
void *do_work_two(void *param)
{
   pthread_mutex_lock(&second_mutex);
   pthread_mutex_lock(&first_mutex);
   /**
    * Do some work
    */
   pthread_mutex_unlock(&first_mutex);
   pthread_mutex_unlock(&second_mutex);

   pthread_exit(0);
}
```

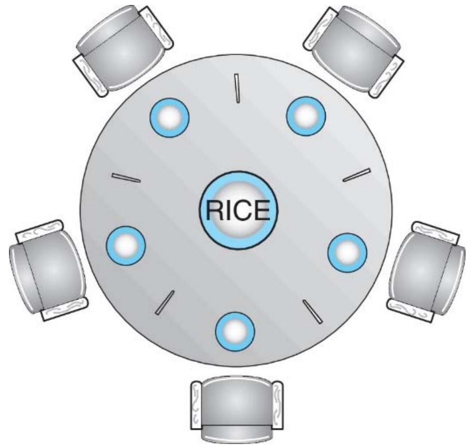
P.326

Handling Deadlocks

- Ensure the system will never enter a deadlock state
  - deadlock prevention: 事前防範ensure that at least one of the 4 necessary conditions cannot hold
  - deadlock avoidance: 邊走邊看 dynamically examines the resource-allocation state before allocation
- Allow deadlocks occur and then recover them
  - deadlock detection
  - deadlock recovery
- Ignore deadlocks
  - used by most operating systems, including LINUX and Windows

P.327

Deadlock Prevention



- Prevent “mutual exclusion”
  - Sharable resources
    - Ex: read-only files
  - Not always possible: some resources are not sharable
- Prevent “hold & wait”:
  - When requesting a resource, it does not hold any resource
  - Protocol1: Pre-allocate all resources before executing
  - Protocol2: Allow requesting resource when it has none 全放掉才能拿
  - Resource utilization is low; starvation is possible

Deadlock Prevention

- Prevent “no preemption”
  - When a process is waiting on a resource, all its holding resources are preempted 在等資源的人所持有的資源隨時都可以被搶
    - e.g. T2 is waiting for R2 and holding R1 : R1 is preemptive

      (T1 → R1 → T2 → R2) T2在等R2，所以它持有的R1會被preempted

      R1 can be preempted and reallocated to T1
  - Applied to resources whose states can be easily saved and restored
    - e.g. CPU registers & memory
  - Cannot easily be applied to physical resources
    - e.g. printers & tape drives

Deadlock Prevention

- Prevent "Circular wait" Idea: 規定process只能按照一定的次序要求資源
  - Impose a total ordering of all resources types
  - A process requests resources in an increasing order
    - Let $R = \{R_0, R_1, \ldots, R_N\}$ be the set of resource types

      $R_i \to T_k \to R_j \Leftrightarrow F(R_i) < F(R_j)$ F(x) → x的order

      如果Tk已要求過Ri，現在要求Rj，則i一定小於j (亦即由小到大要求)
  - Example:
    - F(SSD drive) = 1, F(HDD drive) = 5, F(printer) = 12
    - request order in program: SSD → HDD → printer
  - Proof: assuming circular wait exists, then 先假設circular wait成立

$$T_0 \to R_0 \to T_1 \to R_1 \to T_2 \to R_2 \to \ldots \to T_n \to R_n \to T_0 \to R_0$$

$$F(R_0) < F(R_1) < \ldots < \underbrace{F(R_n) < F(R_0)}_{\text{違反規定!}} \Rightarrow \text{ not possible!}$$

Order Example

```
pthread_mutex_t first_mutex;
pthread_mutex_t second_mutex;

pthread_mutex_init(&first_mutex,NULL);
pthread_mutex_init(&second_mutex,NULL);
```

T1

```
/* thread_one runs in this function */
void *do_work_one(void *param)
{
    pthread_mutex_lock(&first_mutex);
    pthread_mutex_lock(&second_mutex);
    /**
     * Do some work
     */
    pthread_mutex_unlock(&second_mutex);
    pthread_mutex_unlock(&first_mutex);

    pthread_exit(0);
}
```

T2

```
/* thread_two runs in this function */
void *do_work_two(void *param)
{
    pthread_mutex_lock(&second_mutex);
    pthread_mutex_lock(&first_mutex);
    /**
    pthread_mutex_unlock(&second_mutex);

    pthread_exit(0);
}
```

T2要改成先要求first →then second
如此一來T1、T2只能有一個人先搶到first

P.330

Deadlock Avoidance

- 和Prevent的區別: 事前防 vs. 邊走邊閃
  - Prevent: Limit how request can be made (請求的方式)
  - Avoidance: Requires additional information about how resources are to be requested; 若認為有deadlock就reject請求
- Two approaches
  - Single instance resource
    - resource-allocation graph (RAG) algorithm
    - based on circle detection
  - Multiple instances resource
    - banker’s algorithm based on safe sequence detection

RAG Algorithm

Request edge: $T_i \rightarrow R_j$ $\quad$ T_i is waiting for R_j

Assisnment edge: $R_j \rightarrow T_i$ $\quad$ R_j has been allocated to T_i

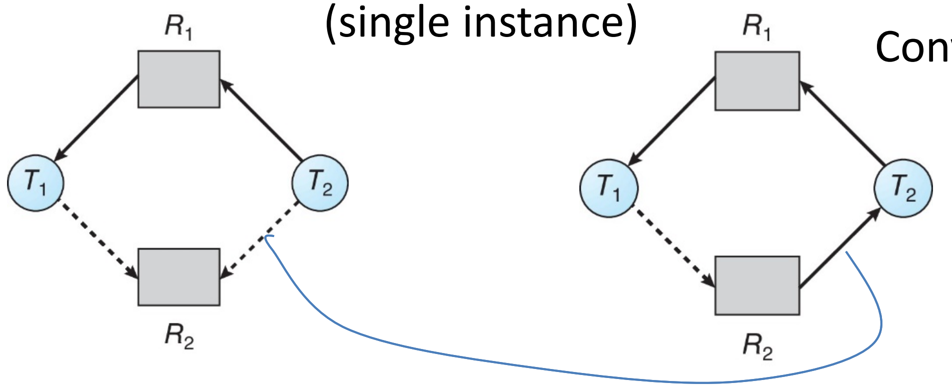
Claim edge: $T_i --> R_j$ $\quad$ T_i may request R_j in the future

- The request can be granted only if converting the claim edges to assignment edges does not result in cycles

直覺意義: 如果未來R配置給T之後形成cycle的話，就不允許此要求

(single instance)

Convert完 →形成cycle

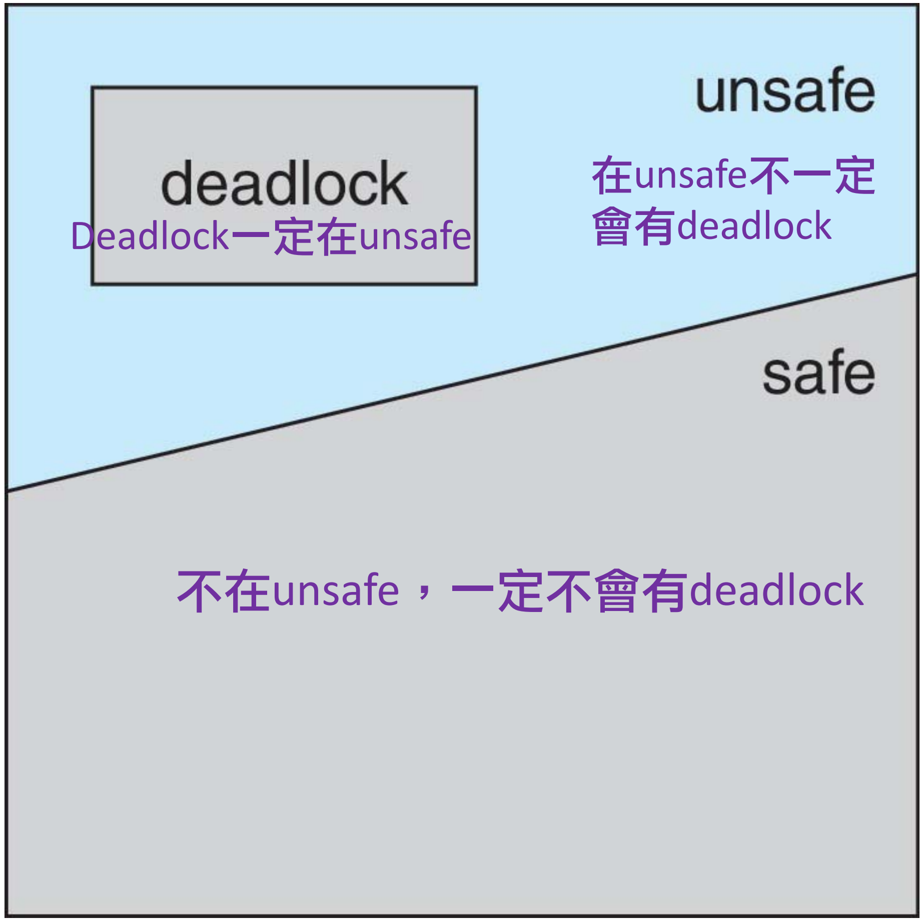


Claim edge to assignment edge

Limitations of RAG

- Single resource
- Resources must be claimed a priori in the system
  - 要事先規畫好，不能臨時多要
- Checking for safety using RAG is O(n^2)

Safe, Unsafe, Deadlock State



Safe State

- Safe state 存在至少一種滿足所有人需求的資源配置次序
  - A system is in a safe state if there exists a sequence of allocations to satisfy requests by all threads
  - This sequence of allocations is called safe sequence
- A system is in a safe state if there exists a sequence

Allocation sequence

$\exists T = < T_1, T_2, \ldots T_n > \rightarrow$

目前沒人用的 (現在可滿足)

$\forall T_i \in T,\ T_i$'s request can be satisified by the free resources or

the resources held by T_j, where $j < i$

j比i早: 有人用，但接下來將釋出的 (未來可滿足)

Safe State with Safe Sequence

- There are 12 resources; 共12個，剩3個
- Assuming at t0: ➔ <T1, T0, T2> is a safe sequence

一般都會考量Worst case (Max needs)

Hint from processes

	Max Needs	Current Holding
T0	10	5
T1	4	2
T2	9	2

(2) T1還需要2個

(3) 3-2=1，可供給!

(1)目前hold了9個resources; 剩12-9=3個resources

(4) T1 結束→ release 4個資源 → 1(原來剩下的)+4 = 5

Safe State with Safe Sequence

- 共12個，剩5個
- Assuming at t0: ➔ <T1, T0, T2> is a safe sequence

Hint from processes →

	Max Needs	Current Holding
T0	10	5
T1	4	2
T2	9	2

(2) T0還需要5個

(3) 5-5=0，可供給!

(1)目前共5個resources

(4) T0 結束→ release 10個資源

Safe State with Safe Sequence

- 共12個，剩10個
- Assuming at t0: ➔ <T1, T0, T2> is a safe sequence

Hint from processes

	Max Needs	Current Holding
T0	10	5
T1	4	2
T2	9	2

(2)目前共10個resources

(3) 10-7=3，**可供給!**

(1)T2**還需要7個**

(4) T2 結束→ release 9個資源;
9+3=12個資源

Un-Safe State without Safe Sequence

目前共hold了10個resources; 12-10=2
T1 Max Needs 2 (4-2=2) → 2-2 = 0，可供給
T1 結束→ release 4 → 總資源4
T0 Max Needs 5 > 4，無法供給!
T2 Max Needs 6 > 4 ，無法供給! (二條路都不通)

- Assuming at t1:

	Max Needs	Current Holding	Available
T0	10	5	
T1	4	2	2
T2	9	~~2~~ 3	

12-(5+2+3)=2

if T2 requests & is allocated 1 more resource

→資源剩2，接下來只能選T1

→No safe sequence exist (T1→ … 二條路都不通)

→ this allocation enters the system into an unsafe state

- A request is only granted if the allocation leaves the system in a safe state

Banker's Algorithm

- Use for multiple instances of each resource type
- Banker algorithm:
  - A general safety algorithm to pre-determine if any safe sequence exists after allocation
  - Only proceed the allocation if safe sequence exists
- Safety algorithm: P.335
  - Assume threads need maximum resources
  - Find a thread that can be satisfied by free resources
  - Free the resource usage of the thread
  - Goto step 2 (until all threads are satisfied)

Banker's Algorithm Example (Safety Algo.)

- Total instances: A:10, B:5, C:7
- Used instances: A:7, B:2, C:5
- Available instances: A:3, B:3, C:2

(3)

	Max			Allocation			Need(Max.-Alloc.) (1)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1
(2)				7	2	5			

3, 3, 2

綠色線代表可行

- Safe sequence: P1 (4)

Banker's Algorithm Example (Safety Algo.)

- Total instances: A:10, B:5, C:7
- Available instances: A:3, B:3, C:2

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

分配給P1資源並執行

3, 3, 2 原來剩下
1, 2, 2 P1還需
2, 1, 0 系統剩餘資源

完成後歸還資源

3, 2, 2 P1歸還
2, 1, 0 系統剩餘資源
5, 3, 2

- Safe sequence: P1

Banker's Algorithm Example (Safety Algo.)

- Total instances: A:10, B:5, C:7
- Available instances: A:5, B:3, C:2

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

分配資源並執行

5, 3, 2 原來剩下
0, 1, 1 P3還需
5, 2, 1 系統剩餘資源

完成後歸還資源

2, 2, 2 P3歸還
5, 2, 1 系統剩餘資源
7, 4, 3

- Safe sequence: P1, P3

Banker's Algorithm Example (Safety Algo.)

- Total instances: A:10, B:5, C:7
- Available instances: A:7, B:4, C:3

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P2	9	0	2	3	0	2	6	0	0
P4	4	3	3	0	0	2	$\boxed{4}$	$\boxed{3}$	$\boxed{1}$

- Safe sequence: P1, P3, P4

分配資源並執行

7, 4, 3 原來剩下
4, 3, 1 P4還需
3, 1, 2 系統剩餘資源

完成後歸還資源

4, 3, 3 P4歸還
3, 1, 2 系統剩餘資源
7, 4, 5

Banker's Algorithm Example (Safety Algo.)

- Total instances: A:10, B:5, C:7
- Available instances: A:7, B:4, C:5

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P2	9	0	2	3	0	2	6	0	0

分配資源並執行

7, 4, 5 原來剩下

6, 0, 0 P2還需

1, 4, 5 系統剩餘資源

完成後歸還資源

9, 0, 2 P2歸還

1, 4, 5 系統剩餘資源

10, 4, 7

- Safe sequence: P1, P3, P4, P2

Banker's Algorithm Example (Safety Algo.)

- Total instances: A:10, B:5, C:7
- Available instances: A:10, B:4, C:7

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	$\boxed{7}$	$\boxed{4}$	$\boxed{3}$

分配資源並執行

10, 4, 7 原來剩下
7, 4, 3 P0還需
3, 0, 4 系統剩餘資源

完成後歸還資源

3, 0, 4 P0歸還
7, 5, 3 系統剩餘資源
10, 5, 7

- Safe sequence: P1, P3, P4, P2, P0

Resource Request

- Total instances: A:10, B:5, C:7
- Available instances: ~~A:3, B:3, C:2~~ ⇨ 2, 3, 0

多要1 多要2

	Max			Allocation				Need(Max-Alloc)			
	A	B	C	A	B	C		A	B	C	
P0	7	5	3	0	1	0		7	4	3	
P1	3	2	2	~~2~~	~~0~~	~~0~~	⇨ 3, 0, 2 (多要1 多要2)	~~1~~	~~2~~	~~2~~	⇨ (3,2,2)-(3,0,2)=(0,2,0)
P2	9	0	2	3	0	2		6	0	0	
P3	2	2	2	2	1	1		0	1	1	
P4	4	3	3	0	0	2		4	3	1	

- If Request (P1) = (1, 0, 2): P1 allocation ➔ 3, 0, 2
  - Enter another safe state (Safe sequence: P1, P3, P4, P0, P2)

Resource Request

- Total instances: A:10, B:5, C:7
- Available instances: ~~A:3, B:3, C:2~~ $\xrightarrow{\text{A:多要3;B多要3}}$ 0, 0, 2

	Max			Allocation			Need(Max-Alloc)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	~~0~~	~~0~~	~~2~~	~~4~~	~~3~~	~~1~~

P4 Allocation ⇨ 3, 3, 2

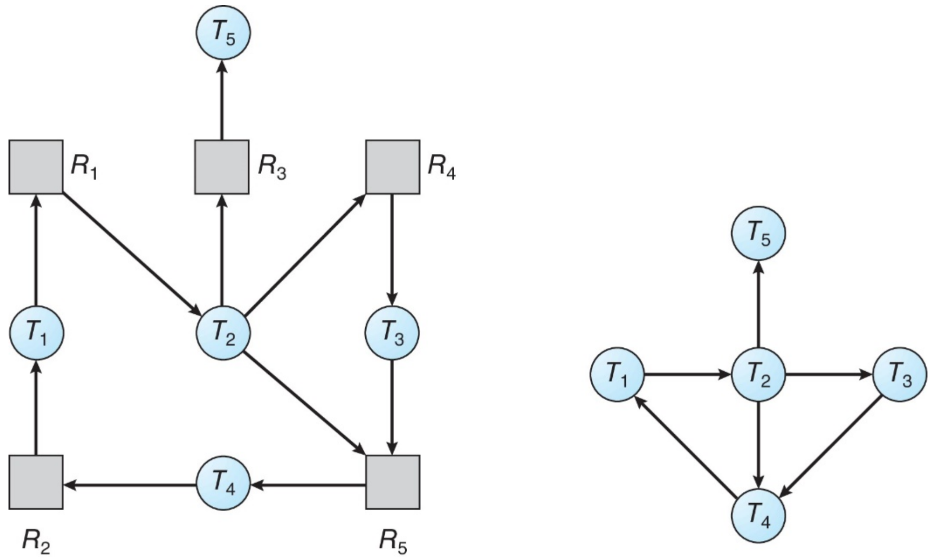
P4 Need ⇨ 1, 0, 1= (4,3,3)-(3,3,2)

每個都不可行!

- If Request (P4) = (3, 3, 0): P4 allocation ➔ 3, 3, 2
  - ➢ enter into an unsafe state (no safe sequence can be found!)

Deadlock Detection

- Single instance 每個時刻都要做
  - convert request/assignment edges into wait-for graph
  - deadlock exists if there is a cycle in the wait-for graph



(a) (b)

Resource-Allocation Graph Corresponding wait-for graph

Multiple-Instance

P.340

- Total instances: A:7, B:2, C:6
- Available instances: A:0, B:0, C:0

	Allocation			Request		
	A	B	C	A	B	C
P0	0	1	0	0	0	0
P1	2	0	0	2	0	2
P2	3	0	3	0	0	0
P3	2	1	1	1	0	0
P4	0	0	2	0	0	2

- The system is in a safe state ➔ <P0, P2, P3, P1, P4>
  ➔ no deadlock
- If P2 request = <0, 0, 1> ➔ no safe sequence can be found
  ➔the system is deadlocked

P.341

Deadlock Recovery

- Process termination
  - Abort all deadlocked processes
  - Abort 1 process at a time until the deadlock cycle is eliminated
    - which process should we abort first? (minimum cost)
- Resource preemption → Many factors, see P.342-343
  - Select a victim: which one to preempt?
  - Rollback: partial rollback or total rollback?
  - Starvation: can the same process be preempted always?

課後閱讀

- P.320 Livelock: 和Deadlock有什麼差別?
- P. 326 Database systems一般用什麼方法來解決 deadlock?
- P.342-343 many factors about process termination and resource preemption

Q & A