Donia said

Ch. 8 : deadlock

Ch.8 → DeadLock

① System model

- Resourse type :- .cpu cycle
  - memory space
  - I/O device
- each process utilize a resource as follows :-
  - request
  - use
  - release

كل process كيف رح يتعامل مع الـ process الجديدة

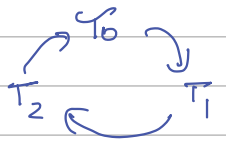
* فال: يطلب مساحة ما ١ ويستخدمها ويحررها.

* Deadlock with Semaphores →

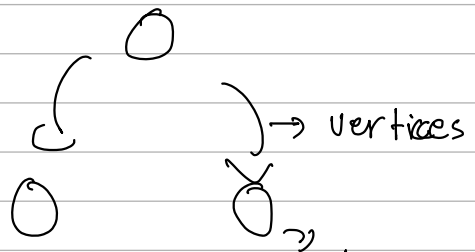
Deadlock Definition :- A Deadlock is a situation which two or more competing actions are each waiting for the other to finish and those neither ever Does.

* four Condition can hold simultaneously arise in Deadlock →
  - Mutul exclusion → عنوع واحدناتي يقوت كل mem. على واحد process
  - Hold and wait
  - No preemption
  - Circular wait

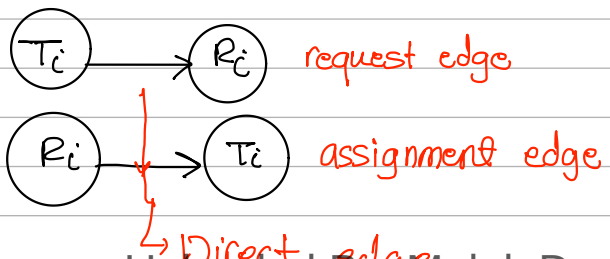
* عالجة الانتظار تتم على شكل دائرة ←



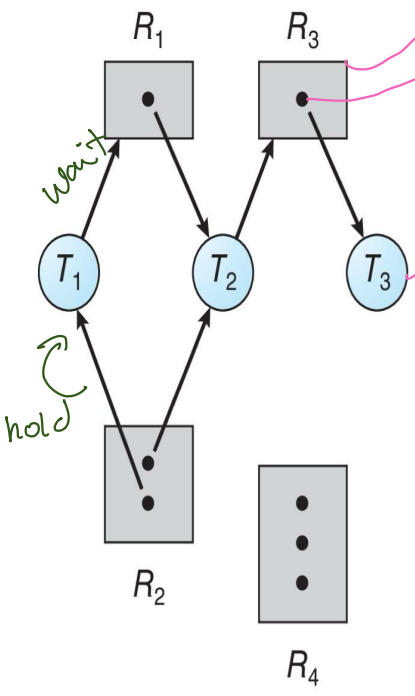
* Resource-Allocation Graph



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



* Resource Allocation Graph example →



Resource instanse

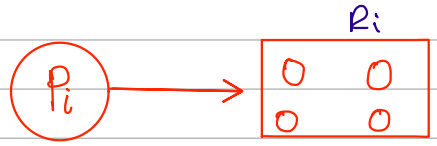
① one instance R_1
② one instance R_2
③ Two instance R_3
④ Three instance R_4

process

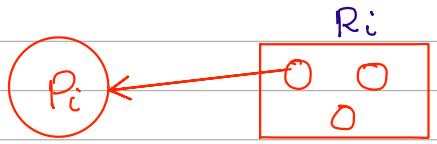
① T_1 hold one instanse of R_2 and waiting for an instance of R_1

② T_2 hold one instanse of R_2 and hold one instance of R_1, is waiting for an instance of R_3

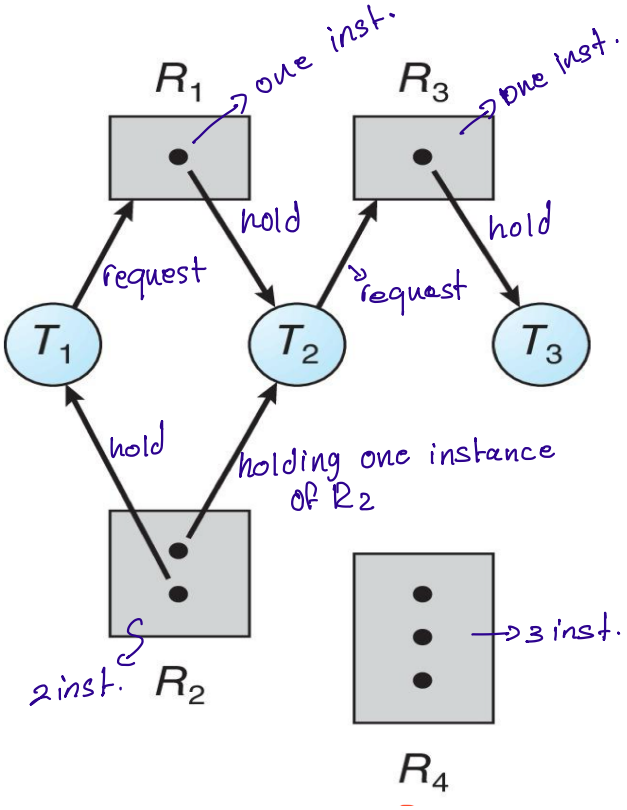
③ T_3 is hold one instance of R_3

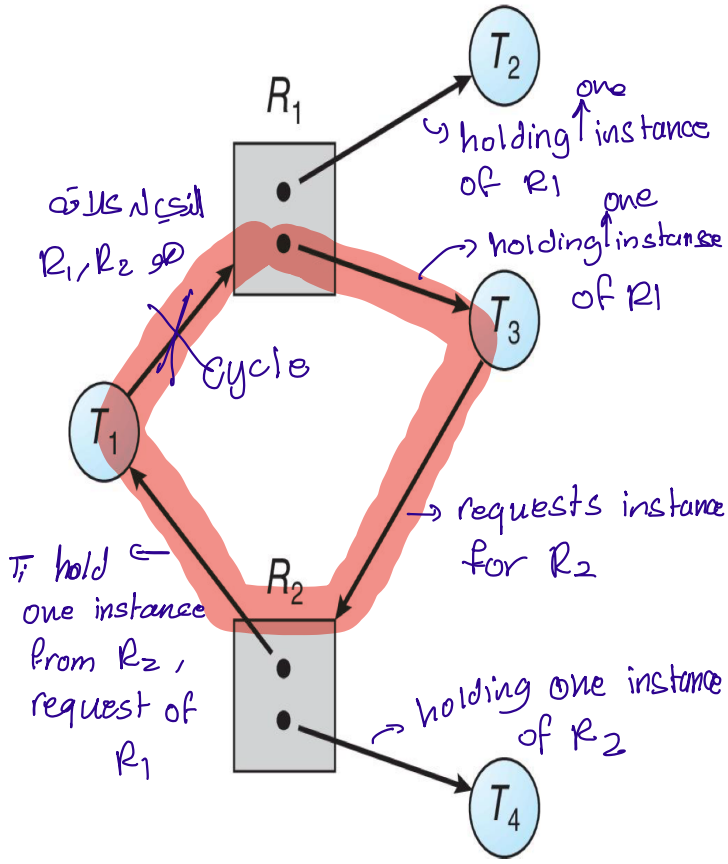


P_i request instance of R_i



P_i holding an instance of R_i





* Basic facts →

Cycle
- ✓ → one inst. → ✓ deadlock
- x → you have several instanst meybe you don't have Deadlock

no cycle → no deadlock

* Method for handling Deadlocks → كيف أتعامل مع الديدلوك
  - Deadlock prevention
  - Deadlock avoidance

* Deadlock prevention :- ① mautual excultion
  ② Hold and wait
  ③ non premption
  ④ Circular wait

عشان امنع حدوث الـ deadlock ← رح اتحكم في عملية الطلب والفرغ.

* DeadLock avoidance →

هاد النظام يكون عنده معلومات جاهزة في النظام

١- يتطلب النموذج الأكثر بساطة وفائدة أن يعلن كل عنصر عن الحد الأقصى لعدد الموارد من كل نوع قد يحتاجه

٢- خوارزمية تجنب الجمود ديناميكيا" حالة تخصيص الموارد للتأكد من عدم وجود حالة انتظار دائرية إذا. (كيف ما يكون عندي Circle-wait condition)

٣- يتم تحديد حالة تخصيص الموارد من خلال عدد الموارد المتاحة والمخصصة والمتطلبات القصوى للعمليات.

يدي	R_1	R_2	R_3
P_1	3	1	0
P_2	4	5	2

كل واحد يأخذ اقصى عدد من الـ R اي موجود عنده

Safe State → الحالة التي يكون عندها النظام آمن

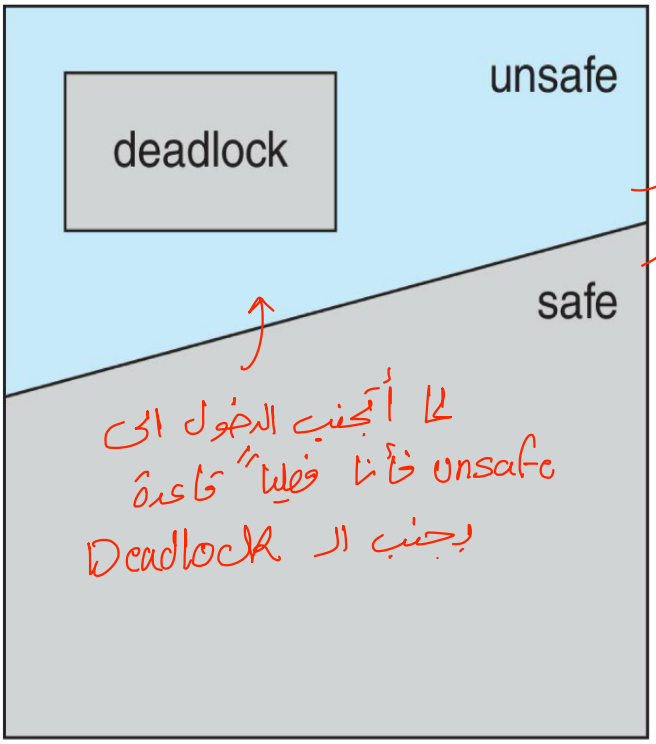
حتى يكون النظام في حالة أمان انه هل اذا أعطيت R_1 شغل و P_2 يكمل عنها ادبلضيها حا يصل

Basic fact →

① if a system is in safe stat → no deadlock

② if a system is in unsafe stat → possibility of deadlock احتمالية

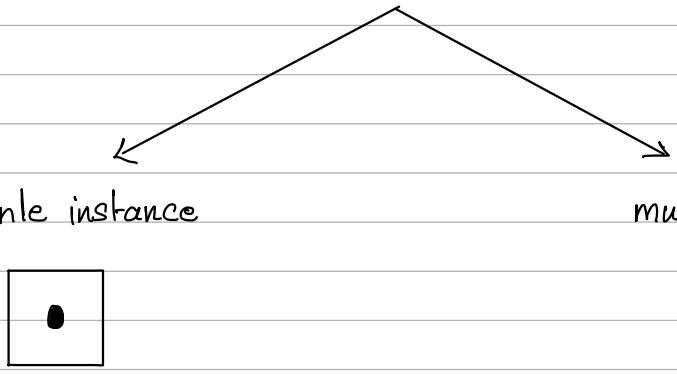
③ Avoidance → ensure that a system will never enter an unsafe state



جزئين

لا أتجنب الدخول الى unsafe خانا فعليا "قاعدة" يجنب الـ Deadlock

Avodance Alogorithms



use a resource allocation graph

use a Banker's algorithm.

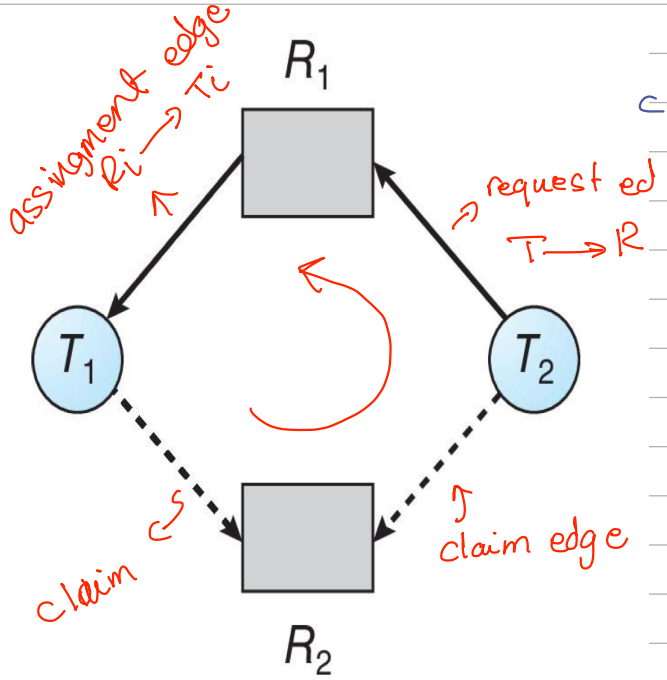
* Resourse-Allocation Graph scheme →

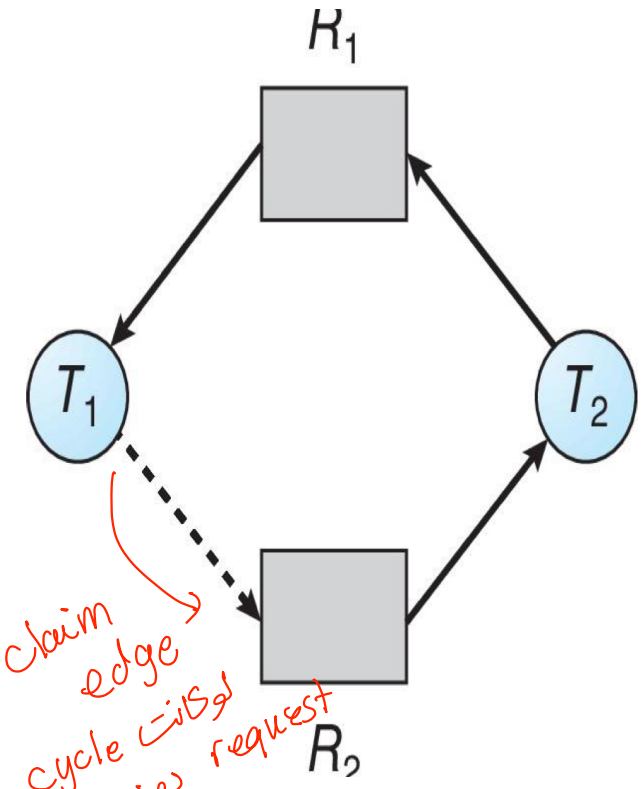
Claim edge → Resource الى process من edge يشير بإتجاه الـ

may request → Pi - - - Ti

Dash line

إذن ممكن يتم طلبه مستقبلاً او لا.





* كيف بدي أمنع النظام يقوت في unsafe edge

g ① Cycle →
request حاجة $T_1 \rightarrow R_1$ خيو مشكلة لأن
request حاجة T_1 رح تكفل في حالة إذا كانت
لـ R_2 .

→ this is in unsafe beacuse you have a probability to happen a unsafe stat

* Banker's algorithm →

① multiple instance of resources
② each thread must a priori claim maximum use
③ when a thread requeste a resources, it may have to wait.
④ when a thread gets all its resourse it must return them in a finite amount of time.

يجب على resource يكتب ان لاك 35 لحتى
تحل النظام وترجع .

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 x m matrix. If $Max[i,j] = k$, then process T_i may request at most k instances of resource type R_j
- **Allocation**: n x m matrix. If Allocation[i,j] = k then T_i is currently allocated k instances of R_j
- **Need**: n x m matrix. If $Need[i,j] = k$, then T_i may need k more instances of R_j to complete its task

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

- 5 threads T_0 through T_4;
  3 resource types:
  A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time t_0:

	Allocation	Max	Available
	A B C	A B C	A B C
T_0	0 1 0	7 5 3	3 3 2
T_1	2 0 0	3 2 2	
T_2	3 0 2	9 0 2	
T_3	2 1 1	2 2 2	
T_4	0 0 2	4 3 3	
	7 2 5		

A (10 instance) , B (5 instance) , C (7 instance)

available 3 3 2

Need = max - allocation

work = available

need

7 4 3

P_0	3 3 2	≥ ?	7 4 3	X
P_1	1 2 2	≤	3 3 2	✓

then work change = 5 3 2 (200 + 332 = 532)

P_2	6 0 0	≤	5 3 2	X
P_3	0 1 1	≤	5 3 2	✓

then work change = 7 4 3 (532 + 211 =)

P_4	4 3 1	≤	7 4 3	✓

then work change = 7 4 5

P_0 [7 | 4 | 3] ≤ [7 | 4 | 5] ✓

745 + 010

then work change = [7 | 5 | 5]

P_2 [6 | 0 | 0] ≤ [7 | 5 | 5] ✓

755 + 302

then work change = [10 | 5 | 7]

the sequence P_1, P_3, P_4, P_0, P_2 the system in the safe stat.

Example: T_1 Request (1,0,2)

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ true

	Allocation	Need	Available
	A B C	A B C	A B C
T_0	~~0 1 0~~ (0 3 0)	~~7 4 3~~ (7 2 3)	2 3 0
T_1	3 0 2	0 2 0	2 1 0
T_2	3 0 2	6 0 0	
T_3	2 1 1	0 1 1	
T_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < T_1, T_3, T_4, T_0, T_2> satisfies safety requirement
- Can request for (3,3,0) by T_4 be granted?
- Can request for (0,2,0) by T_0 be granted?

A B C — P_4 need

① Request = [3, 3, 0]

1- [3, 3, 0] ≤ [4, 3, 1] ✓

2- [3, 3, 0] ≤ [2, 3, 0] ✗

P_4 you must wait because the Source you don't available.

- Request [3,3,0] by T_4

need [4, 3, 1] / ① [3,3,0] ≤ [4,3,1] ✓

② [3, 3, 0] ≤ [2, 3, 0] ✗

P_4 you must wait because it's not avalible.

- Request [0, 2, 0] by T_0 / ① [0, 2, 0] ≤ [7, 4, 3] ✓
need [7, 4, 3] / ② [0, 2, 0] ≤ [2, 3, 0] ✓

③ Avalible - req = [2, 3, 0] - [0, 2, 0] = [2, 1, 0]

④ allocation + Req = [0,1,0] + [0,2,0] = [0,3,0]

⑤ need = [7,4,3] ⇝ [7,4,3] − [0,2,0]

= [7,2,3]

⑥ work = [2,1,0]

P_0 [7,2,3] ≤ [2,1,0] ✗

P_1 [0,2,0] ≤ [2,1,0] ✗

P_2 [6,0,6] ≤ [2,1,0] ✗

P_0 must wait → the system in unsafe stat
(« Dead lock »)

- Five threads T_0 through T_4; three resource types
  A (7 instances), *B* (2 instances), and *C* (6 instances)

- Snapshot at time t_0:

	Allocation	Request	Available
	A B C	A B C	A B C
T_0	0 1 0	0 0 0	0 0 0
T_1	2 0 0	2 0 2	
T_2	3 0 3	0 0 0	
T_3	2 1 1	1 0 0	
T_4	0 0 2	0 0 2	
	7 2 6		

work [0,0,0]
Finish = 0
T_0 → req ≤ work
work [0,1,0]
T_2 → req ≤ work
work [3,1,3]
T_1 → req ≤ work
work [5,1,3]
[3,1,3]
[0,1,0]

- Sequence <T_0, T_2, T_3, T_1, T_4> will result in ***Finish[i] = true*** for all *i*

T_0, T_2, T_1, T_3, T_4

work < [7,2,4]
work [7,2,6]

T_0 → req ≤ work
[0,0,0] ≤ [0,0,0] ✓

Work [0,1,0]

T_1 → [2,0,2] ≤ [0,1,0] ✗

$T_2 \rightarrow [0,0,0] \leq [0,1,0]$ ✓

work $[3,1,3]$

$T_3 \rightarrow [1,0,0] \leq [3,1,3]$ ✓

work $[5,2,4]$

$T_4 \rightarrow [0,0,2] \leq [5,2,4]$ ✓

work $[5,2,6]$

$T_1 \rightarrow [2,0,2] \leq [5,2,6]$ ✓ 010

work $[5,3,6]$

the sequence $\rightarrow T_0, T_2, T_3, T_4, T_1$