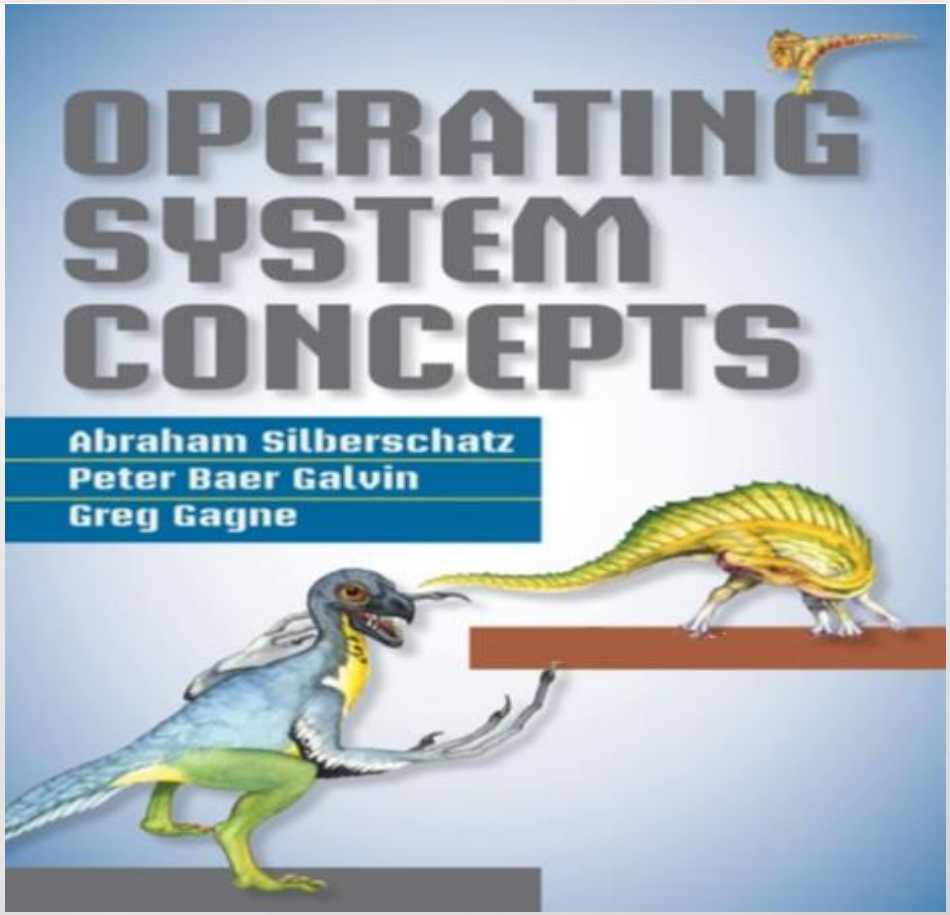
Mustansiriayah University
Collage of Education
Computers Science Department

Chapter Six
Deadlock
Part 4



Fourth Class

Dr. Hesham Adnan ALABBASI

2019-2020

AVOIDANCE ALGORITHMS
BANKER'S ALGORITHM

- Used with Multiple instances of resources
- 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

- تستخدم مع الموارد التي تحوي مثيلات (حالات) متعددة
- يجب على كل Process ان يعلن مسبقاً عن الاستخدام الأقصى من الموارد التي يحتاجها.
- عندما تطلب Process مورد ما قد تضطر إلى الانتظار
- عندما تحصل الـ Process على كل مواردها ، يجب أن تعيدها في فترة زمنية محدودة

Data Structures for the Banker's Algorithm

هيكلة بيانات خوارزمية الـ Banker

Let *n* = number of processes, and *m* = number of resources types.

n تمثل عدد الـ processes, و m تمثل عدد الموارد. الهيكلية تتكون من الأربعة التالية:

- **Available***:* A vector of length ***m*** indicates the number of available resources of each type. If ***Available [j]*** = ***k***, there are ***k*** instances of resource type ***Rj*** are available.

- **Available : (المتوفر),** متجه طوله **m** يشير الى عدد الموارد المتوفرة لكل نوع. اذا كان **Available [j] = k** هذا يعني توفر k من عدد المثيلات (حالات) المصدر **Rj**

- **Max***:* An ***n × m*** matrix defines the maximum demand of each process. I***f Max[i][j]*** =***k***, then process ***Pi*** may request at most ***k*** instances of resource type ***R j***.

- **Max** *(اقصى)*: مصفوفة من **n × m** يعرف الطلب الاقصى لكل process من الموارد.
- اذا كان **Max[i][j] =k** هذا يعني ان الـ **process Pi** ربما يطلب على الاغلب بعدد **k** من حالات المورد **Rj** .

Data Structures for the Banker's Algorithm Cont.

- **Allocation**: ***An n × m matrix*** defines the number of resources of each type currently allocated to each process. If ***Allocation[i][j] = k***, then process ***Pi*** is currently allocated ***k*** instances of resource type ***R j***.

- **Allocation**(المحجوز): مصفوفة من **n × m** تشير الى عدد الموارد المحجوزة من كل نوع لكل process. اذا كان **Allocation[i][j] = k** هذا يعني ان الـ process **Pi** يحجز في الوقت الحالي **k** من حالات المورد Rj.

- **Need**: An ***n × m*** matrix indicates the remaining resource need of each process. If ***Need[i][j] equals k***, then process ***Pi*** may need ***k*** more instances of resource type ***R j*** to complete its task.

- **Need** *(الاحتياج): مصفوفة من* **n × m** *تشير الى عدد الموارد التي يبقى يحتاجها كل نوع لكل process. اذا* **Need[i][j] = k** *هذا يعني ان الـ process* **Pi** *ربما يحتاج الى k من حالات المورد* **Rj** *لكي يكمل عمله*

Need [i][j] = Max [i][j] – Allocation [i][j]

Data Structures for the Banker's Algorithm

- To simplify the presentation of the banker's algorithm, we next establish some notation.
- Let ***X*** and ***Y*** be vectors of length ***n.*** We say that ***X*** ≤ ***Y*** if and only if ***X[i]*** ≤ ***Y[i]*** for all ***i = 1, 2, ..., n.***
- For example, if ***X = (1,7,3,2)*** and *Y* = ***(0,3,2,1)***, then ***Y*** ≤ ***X.*** In addition, *Y* < *X if* *Y* ≤ *X* and *Y* ≠ *X*.

- تبسيط التمثيل للخوارزمية:
- نفرض ***X*** and ***Y*** ***هما متجهين بالطول n . نستطيع القول ان*** ***X*** ≤ ***Y*** فقط فقط في حالة ***X[i]*** ≤ ***Y[i]*** لكل ***i = 1, 2, ..., n.***
- كمثال : كان ***X = (1,7,3,2)*** if *Y* = ***(0,3,2,1)*** و ***اذن X*** ≤ ***Y*** .

Data Structures for the Banker's Algorithm

- We can treat each row in the matrices ***Allocation*** and ***Need*** as vectors and refer to them as ***Allocation i*** and ***Need i*** .
- نستطيع معاملة كل صف في المصفوفات ***Allocation*** and ***Need*** على انهم متجه يشير اليهم بـ ***Allocation i*** و ***Need i*** .

- The vector ***Allocation i*** specifies the resources currently allocated to process ***Pi***
- المتجه ***Allocation i*** *يشير الى الموارد المحجوزة للـ* ***process Pi***.

- The vector ***Need i*** specifies the additional resources that process ***P***i may still request to complete its task.
- المتجه ***Need i*** *يشير الى الموارد الاضافية التي يمكن للـ* ***process Pi*** يبقى يحتاجه لاكمال عمله.

6.5.3.1. Safety Algorithm

We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described, as follows:

هذه الخوارزمية تستخدم في ايجاد هل ان النظام في حالة Safe أو Unsafe

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

لنفرض لدينا المتجه Work بالطول m , والمتجه Finish بالطول n . قيمهم الاولية كم ادناه:

Work = Available and
Finish [i] = false for i = 0, 1, …, n- 1

2. Find an ***i*** such that both:
   (a) ***Finish [i] = false***
   (b) $Need_i \leq Work$
   If no such ***i*** exists, go to step 4
3. $Work = Work + Allocation_i$
   Finish[i] = true
   go to step 2
4. If ***Finish [i] = true*** for all ***i***, then the system is in a safe state

6.5.3.2. Resource-Request Algorithm

We now describe the algorithm which determines if requests can be safely granted.

تستخدم هذه الخوارزمية لتحديد فيما اذا كان الطلب للموارد من اي process يكمن ضمانه او لا.

Let ***Request i*** be the request vector for process ***Pi***.

If ***Request i [j] == k***, then process ***Pi***, wants ***k*** instances of resource type ***Rj.***

When a request for resources is made by process Pi the following actions are taken:

عندما يكون هنالك طلب لموارد من اي process مثلا Pi فنتبع الخطوات التالية:

1. If ***Request i < Need i***, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If ***Request i <= Available***, go to step 3. Otherwise, ***Pi*** must wait, since the resources are not available.
3. Pretend to allocated the requested resources to process ***Pi*** by modifying the state as follows:

Available = Available – Request i

Allocation i= Allocation i + Request i

Need i = Need i- Request i

- If safe ⇒ the resources are allocated to P_i
- If unsafe ⇒ P_i must wait, and the old resource-allocation state is restored

Example of Banker's Algorithm

A system with:
-5 processes P_0 through P_4;
-3 resource types (A, B, C).
-*A* (10 instances), *B* (5instances), and *C* (7 instances)
Snapshot at time T_0:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Answer the following questions using banker algorithm:

1. What is the content of the need matrix?
2. Is the system in safe state or unsafe state? if it safe show the sequences with details?
3. If process P1 request ***one*** additional instance of resource type ***A*** and ***two*** instances of resource type ***C***, so *Request i* = (1,0,2). can the result be granted immediately?

Example (Cont.)

1- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

يتم ايجاد المصفوفة Need باستخدام Max - Allocation

P0 ➔ 753 – 010 = 743, P1 ➔ 322- 200 = 122, …

	Need
	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

2- The system is in a safe state since if we follow this sequence

$$< P_1, P_3, P_4, P_2, P_0 >$$

Which satisfies safety criteria

النظام في حالة Safe اذا اتبعنا التسلسل اعلاه في التنفيذ.

- We can't start with ***P0***, because it needs (**7 4 3**) which is greater than available resources (**3 2 2**).
- لا نستطيع البدء باستخدام P0 لانه يحتاج الى موارد (**7 4 3**) والتي هي اكبر من الموارد المتوفرة (**3 2 2**)

- So, we start with **P1** because Need (1 2 2)<= Available (**332**), P1 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi))

- لذلك, نبدأ بالـProcess P1 لانه يحتاج الى (1 2 2) والتي هي اصغر او تساوي Available (**332**)
- فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available =
3 3 2
2 0 0 +
---
5 3 2

- We can't select **P2** because Need (6 0 0) > new available (**5 3 2**).

- ايضا لا نستطيع اختيار الـ process P2 لان الـ له Need (6 0 0) أكبر من new available (**5 3 2**)
- لذلك نختار **P3 لان له** Need (0 1 1) اصغر او تساوي new available (**5 3 2**).

For this we select **P3** because Need (0 1 1)<= new available (**5 3 2**).
P3 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi))

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available =
532
211 +

743

- Then select **P4** because Need (4 3 1)<= new available (**7 4 3**).
- نختار **P4** لان له Need (4 3 1) اصغر او تساوي new available (**7 4 3**)

P4 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi))

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available = 7 4 3
0 0 2 +
---
7 4 5

- Then select **P2** because Need (6 0 0)<= new available (**7 4 5**).
- نختار **P2** لان له Need (6 0 0) اصغر او تساوي new available (**7 4 5**)
- P2 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi)).

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available = 7 4 5
3 0 2 +
---
10 4 7

الانتباه هنا عند الجمع يتم اعتبار كل مورد كمتجه منفرد عن الاخر فيتم جمعه بصورة معزولة عن المورد الاخر

- Then select **P0** because Need (7 4 3)<= new available (**10 4 7**).
  نختار **P0** لان له Need (743) اصغر او تساوي new available (**10 4 7**)
- P0 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi)).

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available =

```
  10 4 7
   0 1 0+
 ---------
  10 5 7
```

- وبهذه الخطوة وصلنا الى نهاية الحل وتم التأكد من ان التسلسل Sequence التالي :

$< P_1, P_3, P_4, P_2, P_0 >$

يحقق لنا ان النظام في حالة Safe

- ملاحظات للحل:

1- الانتباه هنا عند الجمع يتم اعتبار كل مورد كمتجه منفرد عن الاخر فيتم جمعه بصورة معزولة عن المورد الاخر .
2- قد يكون هنالك اكثر من تسلسل لتنفيذ الـ processes لذلك قد يختلف الحل من طالب الى آخر.
3- للتاكد من ان الحل صحيح نلاحظ الموارد النهائية بعد اكمال الحل. يجب ان تساوي مجموع كل الموارد المحجوزة للـ processes زائد المتوفرة available في النظام قبل البدء بالحل

Note: No. of all resources in the system (Allocation for all resources + Available)

```
  0 1 0
  2 0 0
  3 0 2     +
  2 1 1
  0 0 2
 _____
  7 2 5   مجموع الموارد المحجوزة لكل الـ processes
        +
  3 3 2
 ------
 10 5 7
```

وهو مطابق للناتج النهائي من الموارد الموجودة في النظام بعد الحل

3- If process P1 request ***one*** additional instance of resource type ***A*** and ***two*** instances of resource type ***C***, so *Request i* = (1,0,2).
can the result be granted immediately?
. To decide whether the request of ***P1*** (1,0,2) can be immediately granted, we use Resource-Request Algorithm in section 6.5.3.2.

المطلب الثالث من المثال: اذا طلب الـ process Pi موارد اضافية هي (1,0,2) هل نستطيع تلبية طلبه
في الحال ام لا, غنتبع الخوارزمية الموجودة في المقطع 6.5.3.2

. To decide whether the request of ***P1*** (1,0,2) can be immediately granted, we use Resource-Request Algorithm in section 6.5.3.2.

1-If ***Request i < Need i***, so ***P1*** request (1,0,2) **<** (3,3,2) which is **True**. The go to step 2. If it is false raise an error condition, "the process has exceeded its maximum claim"

يتم فحص الطلب اذا كان اصغر من الـ Need فيتم الانتقال الى الخطوة 2 . واذا لا يظهر له رسالة " لا
يمكن تلبية الطلب لان الـ request اكبر من الـ Need .

2- Check that ***Request*** $\leq$ ***Available*** (that is, (1,0,2) $\leq$ (3,3,2) $\Rightarrow$ **true**, and allocated the requested resources to process ***P1*** by modifying the state as follows:

الخطوة الثانية يتم فحص الطلب اذا كان اصغر او يساوي المتوفر من الموارد (available) فيتم اعطاءه الموارد التي طلبها ويتم تحديث الحالة للنظام كما يلي:

- ***Available = Available – Request I***
- ***Allocation i= Allocation i + Request I***
- ***Need i = Need i- Request i***

Process	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0 (3 3 2-1 0 2)
P1	3 0 2 (2 0 0 + 1 0 2)	0 2 0	
P2	3 0 2	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

الجديد بعد عمل الفقرات اعلاه

We must determine whether this new system state is safe.
To do so, we execute our safety algorithm and find that the sequence $<P_1, P_3, P_4, P_0, P_2>$ satisfies the safety requirement.
Hence, we can immediately grant the request of process P_1.

يجب ان نحدد فيما اذا كان حالة النظام الجديدة هي Safe او Unsafe فنستخدم نفس خوارزمية Safety للتأكد من ان التسلسل $<P_1, P_3, P_4, P_0, P_2>$ يلبي المتطلبات الضرورية .
لذلك يستطيع النظام توفير او منح الموارد الجديدة للـ process الذي طلبها.

Can request for (3,3,0) by P4 be granted?
Can request for (0,2,0) by P0 be granted?

End of Part 4