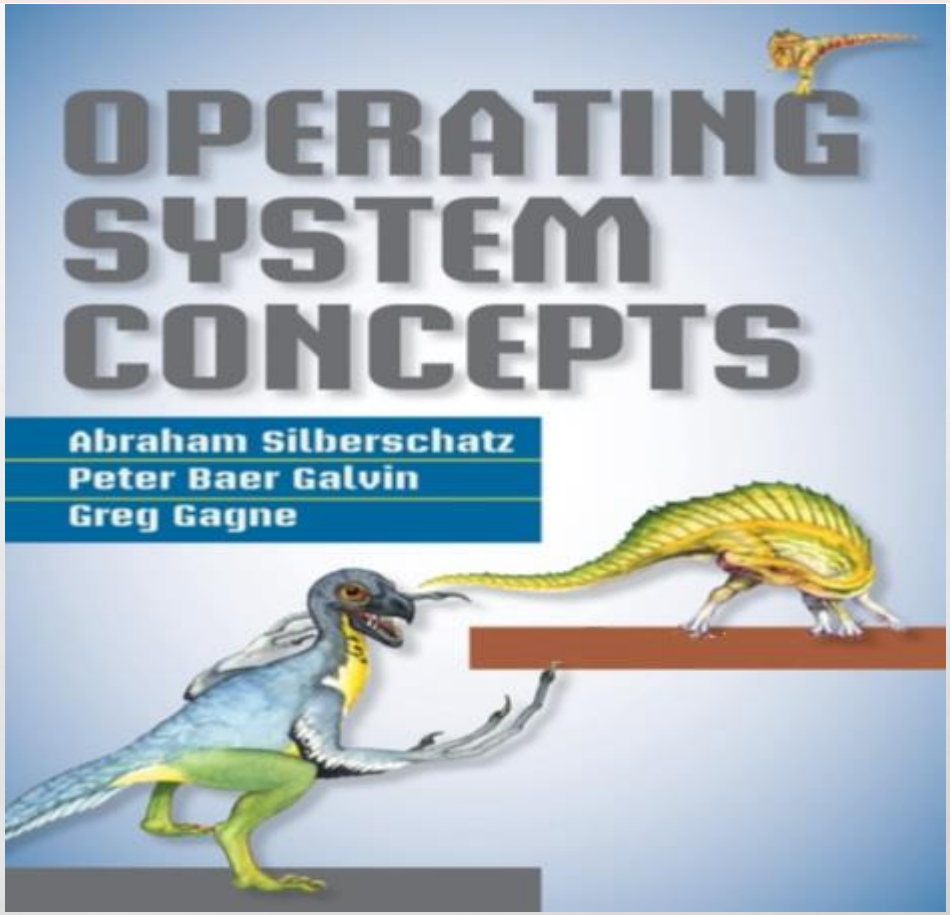
Mustansiriayah University
Collage of Education
Computers Science Department

Chapter Six
Deadlock
Part 2



Fourth Class

Dr. Hesham Adnan ALABBASI

2019-2020

6.3 METHODS FOR HANDLING DEADLOCKS

- We can deal with the deadlock problem in one of three ways:

1. Ensuring that the system will **never** enter a deadlock state.
   - Deadlock **prevention (منع)**
   - Deadlock **avoidance** (تجنب)
2. Allow the system to enter a deadlock state and then recover.
3. Ignore the problem and pretend that deadlocks never occur in the system.

يمكننا التعامل مع مشكلة الdeadlock بإحدى الطرق الثلاث:

1. التأكد من أن النظام لن يدخل إلى حالة الdeadlock .
   - منع الـ deadlock
   - تجنب الـ deadlock
2. السماح للنظام بدخول حالة الـ deadlock ثم معالجته.
3. تجاهل المشكلة وتظاهر بأن الـ deadlock لا يحدث أبداً في النظام.

1- To ensure that deadlocks never occur, the system can use either:

- a deadlock prevention or
- a deadlock avoidance scheme.

- Deadlock prevention provides a set of methods to ensure that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how requests for resources can be made.

يوفر منع حالة الـ deadlock مجموعة من الطرق لضمان عدم توفر واحد على الأقل من الشروط الضرورية .تمنع هذه الطرق الـ deadlock عن طريق تقييد كيفية تقديم طلبات الموارد.

- Deadlock avoidance requires that the operating system be given additional information in advance concerning which resources a process will request and use during its lifetime. With this additional knowledge, the operating system can decide for each request whether or not the process should wait.

يتطلب تجنب الـ Deadlock إعطاء نظام التشغيل معلومات إضافية مقدمًا بشأن الموارد التي ستطلبها وتستخدمها الـ process خلال فترة حياتها .من خلال هذه المعرفة الإضافية ، يمكن لنظام التشغيل أن يقرر لكل طلب ما إذا كانت الـ process ستنتظر أم لا

To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

لتحديد ما إذا كان يمكن تلبية الطلب الحالي أو تأجيله ، يجب على النظام مراعاة الموارد المتاحة حاليًا ، والموارد المخصصة حاليًا لكل process، والطلبات والإصدارات المستقبلية لكل process .

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may arise. In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock (if a deadlock has indeed occurred).

إذا كان النظام لا يستخدم إي من طرق منع أو تجنب حالة الـ deadlock ، فقد ينشأ أو يحدث الـ deadlock

في هذه البيئة ، يمكن للنظام توفير خوارزميات:

- تفحص حالة النظام لتحديد ما إذا كان قد وصل إلى الـ deadlock
- وخوارزمية لمعالجة او حل الـ deadlock (إذا حدث بالفعل الـ deadlock).

In the absence of algorithms to detect and recover from deadlocks, we may arrive at a situation in which the system is in a deadlocked state yet has no way of recognizing what has happened.

في حالة عدم وجود خوارزميات للكشف عن حالات الـ deadlock والتعافي منها ، قد نصل إلى وضع يكون فيه النظام في حالة الـ deadlock ولكن لا توجد طريقة للتعرف على ما حدث.

6.4. DEADLOCK PREVENTION

■ For a deadlock to occur, each of the four necessary conditions must hold. **By ensuring that at least one of these conditions cannot hold**, we can **prevent** the occurrence of a deadlock. We elaborate on this approach by examining each of the four necessary conditions separately.

من أجل حدوث الـ deadlock ، يجب تحقق كل من الشروط الأربعة الضرورية معاً .
ومن خلال التأكد من أن شرطًا واحدًا على الأقل من هذه الشروط لا يمكن أن يحدث ، يمكننا منع حدوث الـ deadloc .
نقوم بتوضيح هذا النهج من خلال فحص كل من الشروط الأربعة الضرورية بشكل منفصل.

- **6.4.1 MUTUAL EXCLUSION:** only one process at a time can use a resource. The mutual-exclusion condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes. Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are essentially non-sharable.

الاستبعاد المتبادل: يمكن ل process واحدة فقط في كل مرة استخدام المورد. يجب أن يكون شرط الاستبعاد المتبادل ساريًا على الموارد غير القابلة للمشاركة .على سبيل المثال ، لا يمكن مشاركة الطابعة في وقت واحد من خلال عدة processes .وعلى النقيض من ذلك ، لا تتطلب الموارد القابلة للمشاركة الوصول الحصري المتبادل وبالتالي لا يمكن أن تشارك في الـ deadlock .بشكل عام ، ومع ذلك ، لا يمكننا منع ال deadlock من خلال رفض شرط الاستبعاد المتبادل ، لأن بعض الموارد غير قابلة للمشاركة بشكل أساسي.

6.4.2. Hold and Wait: To ensure that the hold-and-wait condition, never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.

- One protocol that can be used requires **each process to request and be allocated all its resources before it begins execution**.
- An alternative protocol **allows a process to request resources only when it has none.** A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

للتأكد من أن شرط **Hold and Wait** ، لا يحدث أبدًا في النظام ، يجب أن نضمن أنه ، كلما طلبت الـ process مورد ما ، فإنها لا تحتفظ بأي موارد أخرى.

- بروتوكول واحد يمكن استخدامه يتطلب من كل process ان يطلب وتخصيص جميع مواردها قبل بدء التنفيذ.
- بروتوكول بديل يسمح للـ process بطلب الموارد فقط عندما لا يكون لديه أي مورد آخر. قد تطلب الـ process بعض الموارد وتستخدمها . قبل أن يتمكن من طلب أي موارد إضافية ، يجب عليه الإفراج عن جميع الموارد التي يتم تخصيصها حاليًا.

ملاحظة: قراءة الامثلة الموجودة في المحاضرات

6.4.3. NO PREEMPTION

- The third necessary condition for deadlocks is that there is **no preemption of resources that have already been allocated**. To ensure that this condition does not hold, we can use the following protocol.
- If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- The released resources (preempted) are added to the list of resources for which the process is waiting.
- The process will be restarted only when it can regain (استرجاع)its old resources, as well as the new ones that it is requesting.

الشرط الثالث الضروري للخروج من الطريق المسدود هو أنه لا يوجد استباق (قطع) للموارد التي تم تخصيصها بالفعل .لضمان عدم استمرار هذا الشرط ، يمكننا استخدام البروتوكول التالي.

- إذا كانت الـ process تحتفظ ببعض الموارد وتطلب موردًا آخر لا يمكن تخصيصه لها على الفور ، فسيتم تحرير جميع الموارد المحتفظ بها حاليًا.
- تضاف الموارد المفرج عنها (المستقطعة) إلى قائمة الموارد التي تنتظرها الـ process .
- سيتم إعادة تشغيل الـ process فقط عندما تتمكن من استعادة (استرجاع) مواردها القديمة ، بالإضافة إلى الموارد الجديدة التي تطلبها.

6.4.4. CIRCULAR WAIT: One way to ensure that this condition never holds is to impose (فرض) a total ordering of all resource types and require that each process requests resources in an increasing order of enumeration

إحدى الطرق للتأكد من عدم استمرار هذا الشرط أبدًا هي فرض ترتيب كامل لجميع أنواع الموارد ويتطلب من كل process بطلب موارد ان يكون بترتيب متزايد للعد.

To illustrate, we let *R = {R1, R2, ..., Rm}* be the set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering.

للتوضيح ، نفرض *R = {R1, R2, ..., Rm}* هي مجموعة أنواع الموارد لكل نوع من الموارد يتم تعيين رقمًا صحيحًا فريدًا ، مما يسمح لنا بمقارنة مصدرين وتحديد ما إذا كان أحدهما يسبق آخر في الترتيب.

We define a one-to-one function *F: R→N*, where *N* is the set of natural numbers. For example, if the set of resource types *R* includes tape drives, disk drives, and printers, then the function F might be defined as follows:

يتم تعريف دالة one-to-one (F: R → N) ، حيث N هي مجموعة الأعداد الطبيعية. على سبيل المثال ، إذا كانت مجموعة أنواع الموارد R تتضمن محركات الأشرطة ومحركات الأقراص والطابعات, فان الدالة F تكون:

F (tape drive) = 1

F (disk drive) = 5

F (printer) = 12

We can now consider the following protocol to prevent deadlocks: **Each process can request resources only in an increasing order of enumeration**. That is, a process can initially request any number of instances of a resource type —say, *Ri.* After that, the process can request instances of resource type *Rj* if and only if *F(Rj) > F(Ri).* For example, using the function defined previously, a process that wants to use the tape drive and printer at the same time must first request the tape drive and then request the printer. .

يمكننا الآن النظر في البروتوكول التالي لمنع حالات الـ deadlocks : **يمكن لكل process طلب الموارد فقط بترتيب متزايد من التعداد** .أي أن الـ process يمكن أن تطلب مبدئيًا أي عدد من حالات نوع المورد - مثل Ri . بعد ذلك ، يمكن أن تطلب الـ process مثيلات من نوع المورد Rj إذا وفقط إذا كان F (Rj)> F (Ri) على سبيل المثال ، باستخدام الوظيفة المحددة مسبقًا ، الـ process التي تريد استخدام محرك الأشرطة والطابعة في نفس الوقت يجب ان تطلب محرك الأشرطة أولاً ثم طلب الطابعة

Alternatively, we can require that a process requesting an instance of resource type *Rj* must have released any resources *Ri* such that *F(Ri) ≥ F(Rj).* Note also that if several instances of the same resource type are needed, a single request for all of them must be issued.
If these two protocols are used, then the circular-wait condition cannot hold

بدلاً من ذلك ، يمكننا أن نحدد أن الـ process الذي يطلب مثيل من نوع المورد Rj يجب أن تكون قد أطلق أي موارد مثلRi . بحيث يكون F (Ri) ≥ F (Rj) . لاحظ أيضًا أنه في حالة الحاجة إلى عدة حالات من نفس نوع المورد ، يجب إصدار طلب واحد لهم جميعًا.
إذا تم استخدام هذين البروتوكولين ، فلا يمكن أن يستمر شرط الـ circular-wait

End of Part 2