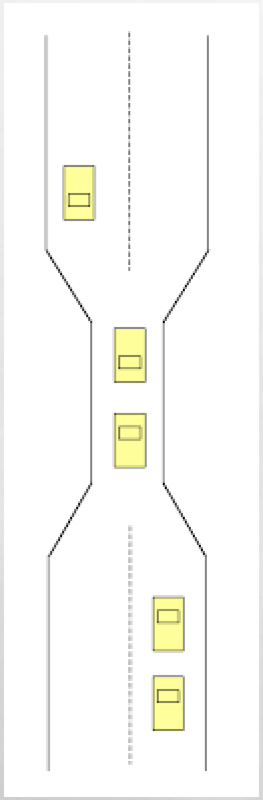
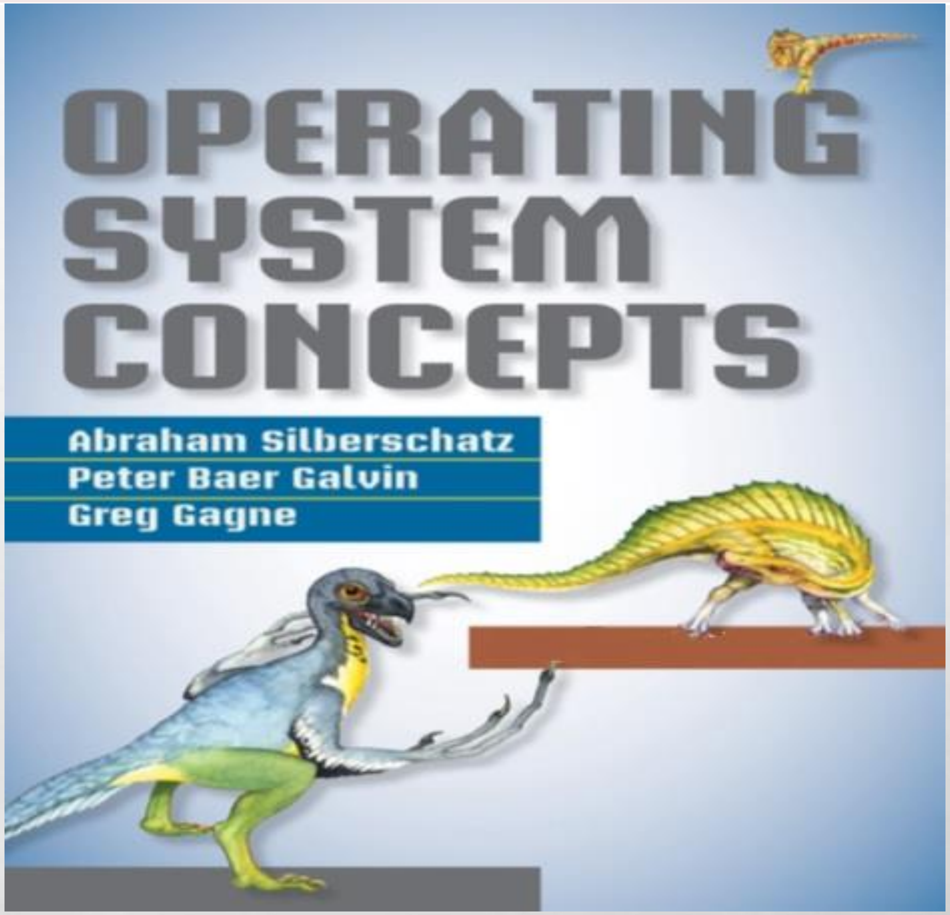
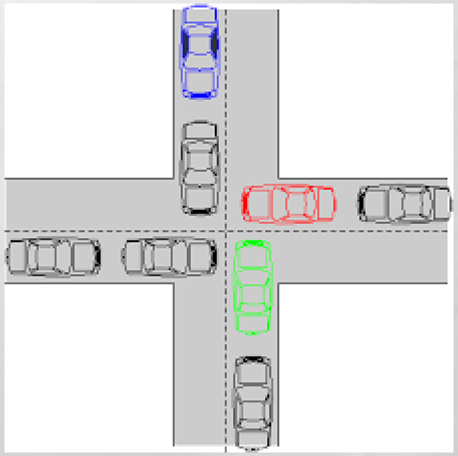
Mustansiriayah University
Collage of Education
Computers Science Department

Chapter Six Deadlock Part I

Fourth Class







Dr. Hesham Adnan ALABBASI

2019-2020

6.1.Deadlock

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state.

  في بيئة البرمجة المتعددة ، قد تتنافس عدة processes على عدد محدود من الموارد. الـ processesتطلب الموارد؛ إذا كانت الموارد غير متوفرة في ذلك الوقت ، فإن ال process تدخل حالة انتظار.

- Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.
- في بعض الأحيان ، لا يمكن لل process التي تنتظر ان تغير حالتها ، لأن الموارد التي طلبتها يتم الاحتفاظ بها بواسطة waiting processes أخرى. تسمى هذه الحالة Deadlock

- operating systems typically do not provide deadlock-prevention facilities, and it remains the responsibility of programmers to ensure that they design deadlock-free programs.
- لا توفر أنظمة التشغيل عادةً تسهيلات منع الDeadlock ، ويبقى من مسؤولية المبرمجين التأكد من تصميم برامج خالية من الDeadlock .

6.1.1. System Model

- System consists of a finite number of resources
- Resource types R1, R2, . . ., Rm

  CPU cycles, memory space, I/O devices
- Each resource type Ri has Wi instances.

- A process must request a resource before using it and must release the resource after using it.
  - يجب ان يطلب الـ process الموارد قبل استخدامها ويجب ان يحررها بعد استخدامها

- A process may request as many resources as it requires to carry out its designated task. The number of resources requested may not exceed the total number of resources available in the system.
  - قد تطلب الprocess ما تحتاج إليه من موارد لتنفيذ المهمة المحددة لها. بحيث لا يتجاوز عدد الموارد المطلوبة عن عدد الموارد المتاحة في النظام.

System Model Cont.

- Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. **Request:** The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

- تطلب الـ process المورد. إذا كان لا يمكن منح الطلب على الفور (على سبيل المثال ، إذا كان المورد قيد الاستخدام من قبل process أخرى) ، فيجب أن تنتظر process التي قامت بالطلب حتى تتمكن من الحصول على المورد.

2. **Use:** The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

- تستطيع الـ process أستخدام المورد. (كمثال: اذا كان المورد هو printer , فتستطيع الـ process الطباعة على هذه الـ Printer .

3. **Release:** The process releases the resource.

- الـ process يحرر المورد

To illustrate a deadlocked state:

- consider a system with **three CD R/W drives**. Suppose each of **three processes** holds one of these CD R/W drives.

➢ If each process is now requesting another drive, the three processes will be in a deadlocked state.

➢ Each is waiting for the event "**CD RW is released**," which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.

- لتوضيح حالة الDeadlock :

- النظام به مورد واحد من الـ محركات أقراص CD R / W. به ثلاثة من هذا النوع مع ثلاثة processes.

- افترض أن كل process من الثلاث يستخدم او يمسك أحد محركات أقراص CD R / W.

- إذا كانت كل process تطلب الآن محرك أقراص آخر ، فستكون ال processes الثلاث في حالة Deadlock.

- كل منها ينتظر الحدث "تم تحرير CD RW" والذي يمكن أن يحدث فقط بسبب إحدى waiting processes الأخرى. يوضح هذا المثال حالة Deadlock لنظام يتضمن نفس نوع المورد.

- Deadlocks may also involve different resource types.
- For example, consider a system with one **printer,** and one **DVD drive**.
- Suppose that process ***Pi*** is holding the DVD and process ***Pj*** is holding the printer.
- If ***Pi*** requests the printer and ***Pj*** requests the DVD drive, a deadlock occurs.

- قد تشمل الDeadlock أيضا أنواع مختلفة من الموارد.

على سبيل المثال ، النظام به طابعة واحدة ومحرك أقراص DVDواحد.
افترض أن ***Pi*** process يستخدم او يمسك قرص DVD وأن
Pj process ت يستخدم او يمسك الطابعة.
إذا طلبت Pi الطابعة وطلبت Pj محرك أقراص DVD، يحدث
.Deadlock

6.2. Deadlock Characterization

In a deadlock, processes never finish executing, and system resources are tied up (مقيد), preventing other jobs from starting

6.2.1. Necessary Conditions

Deadlock can arise if four conditions hold simultaneously (في نفس الوقت).

1. **Mutual exclusion**: At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

   الاستبعاد المتبادل: يجب الاحتفاظ بمورد واحد على الأقل في وضع غير قابل للمشاركة ؛ بمعنى ، process واحدة فقط في كل مرة يمكن ان يستخدام المورد. إذا طلبت process أخرى ذلك المورد ، فيجب تأجيل الـ process الطالب حتى يتم تحرير المورد.

2. **Hold and wait**: a process holding at least one resource and waiting to acquire additional resources held by other processes

   اللامساك والانتظار: الـ process يمسك موردًا واحدًا على الأقل وتنتظر الحصول على موارد إضافية تمسك بها الـ processes الأخرى

Deadlock Characterization Cont.

3. **No preemption**: Resources cannot be preempted; that is, a resource can be released only by the process holding it, after that process has completed its task.

لا يوجد استباق: لا يمكن استباق (استقطاع) الموارد ؛ أي أنه لا يمكن تحرير المورد إلا من خلال الـ process التي تحتفظ بها ، بعد أن تكمل هذه الـ process مهمتها.

4. **Circular wait**: There exists a set {P0, P1, ..., Pn} of waiting processes, such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ..., Pn-1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.

يوجد على الاقل مجموعة من الـ waiting processes $\{P_0, P_1, \ldots, P_n\}$, P_0 ينتظر مورد يمسك به P_1 , P_1 ينتظر مورد يمسك به P_2 , P_{n-1} ينتظر مورد يمسك به P_n , P_n ينتظر مورد يمسك به P_0 .

6.2.2. Resource-Allocation Graph

Deadlocks can be described more precisely in terms of a directed graph called a system **resource-allocation graph** consists of:

- This graph consists of a set of vertices **V** and a set of edges **E**.

- The set of vertices **V** is partitioned into two different types of nodes:

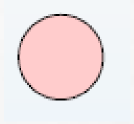
  $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system

  $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system

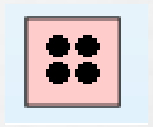
- **Request edge:** A directed edge from process ***Pi*** to resource type ***Rj*** is denoted by Pi → Rj; it signifies that process Pi, has requested an instance of resource type Rj, and is currently waiting for that resource.

- **Assignment edge :** **edge** from resource type ***Rj*** to process ***Pi*** is denoted by Rj → Pi; it signifies that an instance of resource type ***Rj*** has been allocated to process ***Pi***

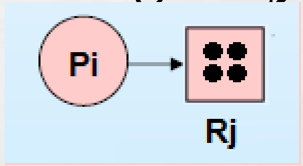
•We represent each process ***Pi***, as a circle



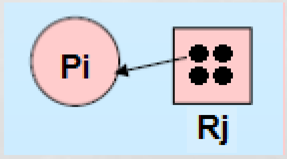
•We represent each resource type ***Rj*** as a rectangle. Since resource type ***Rj*** may have more than one instance, we represent each such instance as a dot within the rectangle.



•Note that a request edge points to only the rectangle ***Rj***,



•An assignment edge must also designate one of the dots in the rectangle (instance).



- When process ***Pi***, requests an instance of resource type ***R***j, a **request edge** is inserted in the resource-allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an **assignment edge**.
- When the process no longer needs access to the resource, it releases the resource; as a result, the assignment edge is deleted.

RESOURCE ALLOCATION GRAPH EXAMPLE

The resource-allocation graph shown in the figure 6.1 depicts the following situation

- The sets P, R, and E:

- P = {PI, P2, P3}

- R = {RI, R2 ,R3, R4}

- E = {PI →RI, P2 → R3, RI → P2, R2 → P2, R2 → PI, R3 →P3 }

- Resource instances:

  - One instance of resource type RI
  - Two instances of resource type R2
  - One instance of resource type R3
  - Three instances of resource type R4

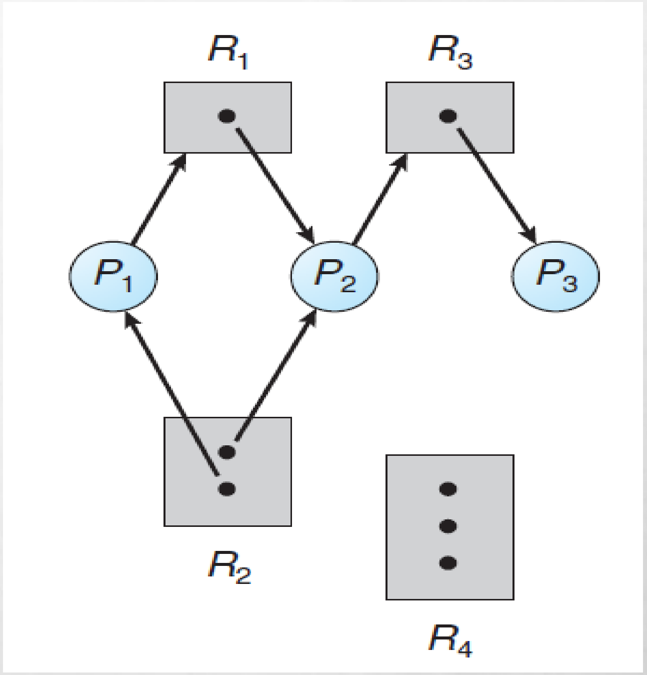


Figure 6.1

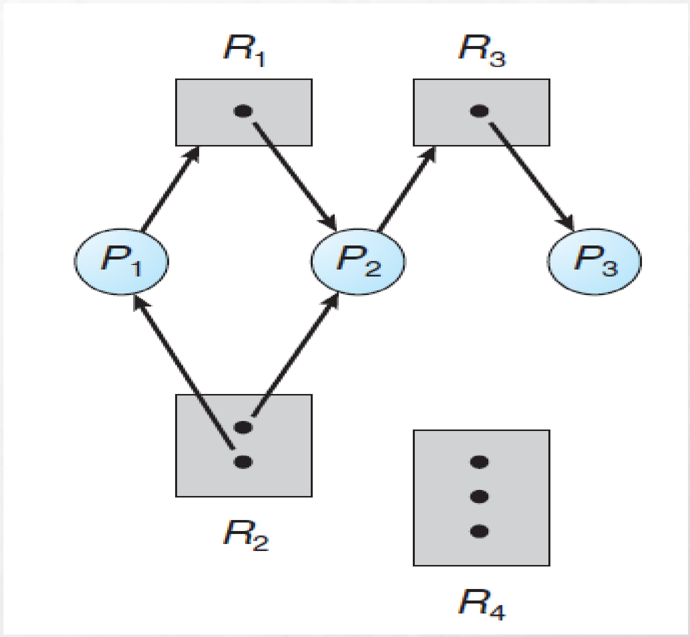
- Process states:

- Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.
- Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.
- Process P3 is holding an instance of R3.

Given the definition of a resource-allocation graph, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. Each process involved in the cycle is deadlocked.

If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred.



من الشكل اعلاه يمكن ملاحظة:
- إذا لم يحتوي الرسم على cycles ، فلن تكون هناك process في النظام في حالة الـ Deadlock .
إذا كان الرسم يحتوي على cycles ، فقد يكون هناك Deadlock.
- إذا كان لكل نوع من الموارد حالة واحدة بالضبط one instance ، فإن الcycle تشير إلى حدوث Deadlocked. كل process تشارك في ال cycle وصلت إلى Deadlock.
- إذا كان لكل نوع من الموارد عدة حالات several instances ، فإن الcycle لا تعني بالضرورة حدوث Deadlock

EXAMPLE 1: To illustrate this concept, we return to the resource-allocation graph depicted in Figure 6.1.

- Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3 → R2 is added to the graph (Figure 6.2)

نفرض ان الـ process P3 تطلب عدد واحد من المورد R2 . وبما ان لا يتوفر في هذه اللحظة هذا المورد (محجوز من PROCESSES) اخرى, فيتم اضافة request edge P3 → R2 كما في الى الشكل 6.2

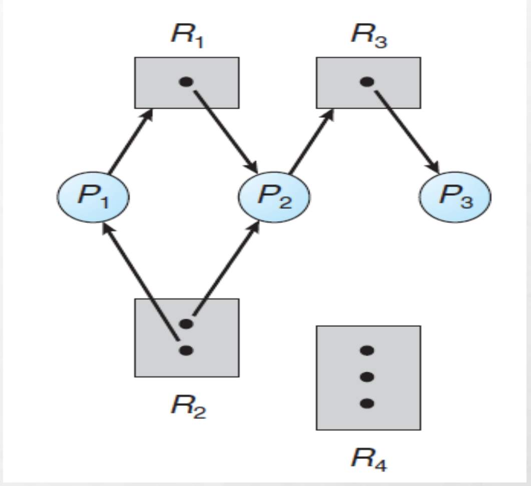


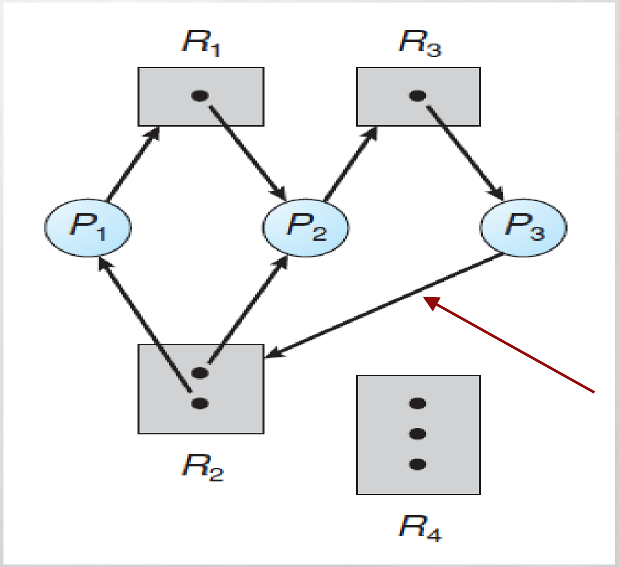
Figure 6.1

- At this point, two minimal cycles exist in the system:

1- P1 →R1→ P2 →R3→ P3 →R2→ P1

2. P2 →R3→ P3 →R2→ P2

Processes P1, P2, and P3 are Deadlocked.

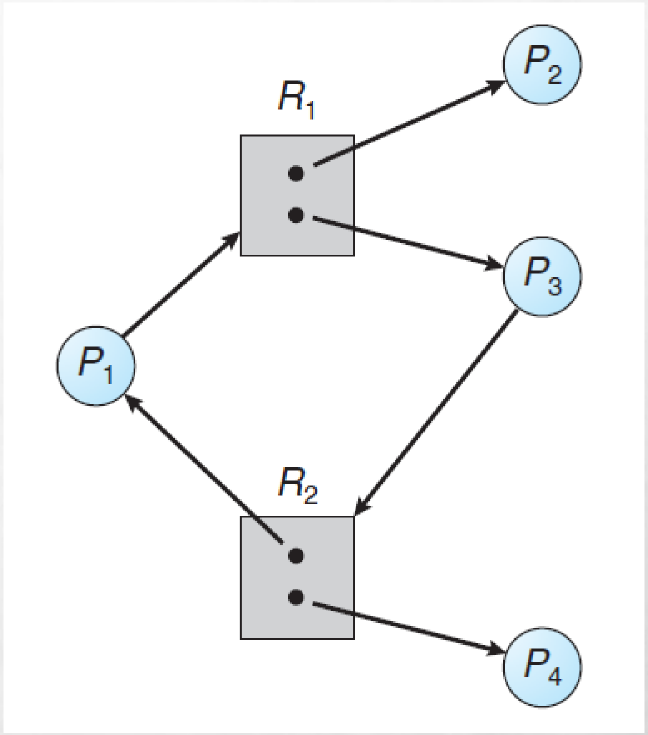


- Process P2 is waiting for the resource R3, which is held by process P3.
- Process P3 is waiting for either process P1 or process P2 to release resource R2.
- In addition, process P1 is waiting for process P2 to release resource R1.

EXAMPLE 2

- Now consider the resource-allocation graph in Figure 6.3. In this example, we also have a cycle: **P1 →R1→ P3 →R2→ P1**
- However, there is no deadlock. Observe that process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.

هنا لا يوجد Deadlock. لاحظ ان الـ process P4 ممكن ان تحرر المورد R2 , وهذا المورد ممكن ان يحجز الى P3 فيتم كسر الـ cycle.



BASIC FACTS

- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock

End of Part1