HỆ ĐIỀU HÀNH
(OPERATING SYSTEM)

Trình bày:Nguyễn Hoàng Việt
Khoa Công Nghệ Thông Tin
Đại Học Cần Thơ

Chương 6: Khoá chết (Deadlock)

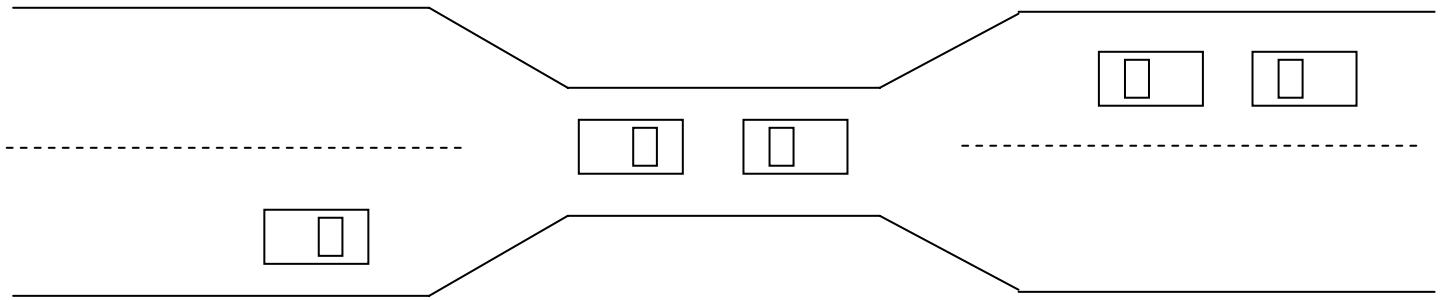
- Vấn đề deadlock
- Mô hình hệ thống
- Tính chất của deadlock
- Các biện pháp khống chế Deadlock
- Ngăn chặn Deadlock
- Tránh Deadlock
- Phát hiện Deadlock
- Phục hồi từ Deadlock
- Giải pháp tích hợp để khống chế Deadlock

Vấn đề Deadlock

- Một tập hợp các quá trình bị nghẽn, mỗi quá trình đang giữ một tài nguyên và cũng đang chờ để xin một tài nguyên khác, mà tài nguyên này lại đang bị giữ bởi một quá trình khác trong tập hợp trên.
- Ví dụ
  - Hệ thống có 2 ổ chứa băng từ.
  - P_1 và P_2, một quá trình đang giữ một ổ và đang cần ổ kia.
- Ví dụ mô phỏng sử dụng semaphore
  - Các semaphores *A* và *B*, khởi tạo là 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

Ví dụ về cầu vượt



- Chỉ có một đường để vượt ngang cầu.
- Một bên của chiếc cầu có thể coi như là một tài nguyên.
- Nếu deadlock xảy ra, có thể giải quyết bằng cách lùi một chiếc xe lại (trưng dụng tài nguyên ra và quay lại (rollback)).
- Có thể sẽ có vài chiếc xe phải lùi lại nếu deadlock xảy ra.
- Sự đói CPU có thể xảy ra.

Mô hình hệ thống (1)

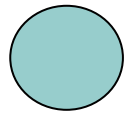
- Kiểu tài nguyên R_1, R_2, . . ., R_m
  - *Ví dụ: CPU cycles, memory space, I/O devices*
- Mỗi tài nguyên R_i có W_i thể hiện.
- Mỗi quá trình sử dụng một tài nguyên như sau:
  - Yêu cầu (request)
  - Sử dụng (use)
  - Giải phóng (release)

Mô hình hệ thống (2)
Đồ thị cấp phát tài nguyên

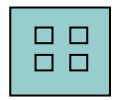
- Bao gồm tập hợp các đỉnh *V* và tập hợp các cạnh *E*.
- *V* được chia làm 2 dạng:
  - $P = \{P_1, P_2, \ldots, P_n\}$, là tập hợp các quá trình đang tồn tại trong hệ thống.
  - $R = \{R_1, R_2, \ldots, R_m\}$, là tập hợp các tài nguyên đang tồn tại trong hệ thống.
- *E* chia làm 2 dạng:
  - Cạnh yêu cầu: cạnh có hướng $P_i \rightarrow R_j$
  - Cạnh cấp phát: cạnh có hướng $R_j \rightarrow P_i$

Mô hình hệ thống (3)
Đồ thị cấp phát tài nguyên

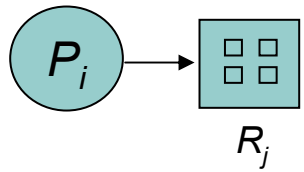
- Quá trình



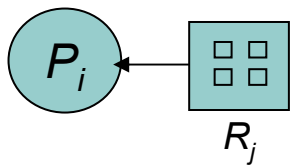
- Tài nguyên với 4 thể hiện



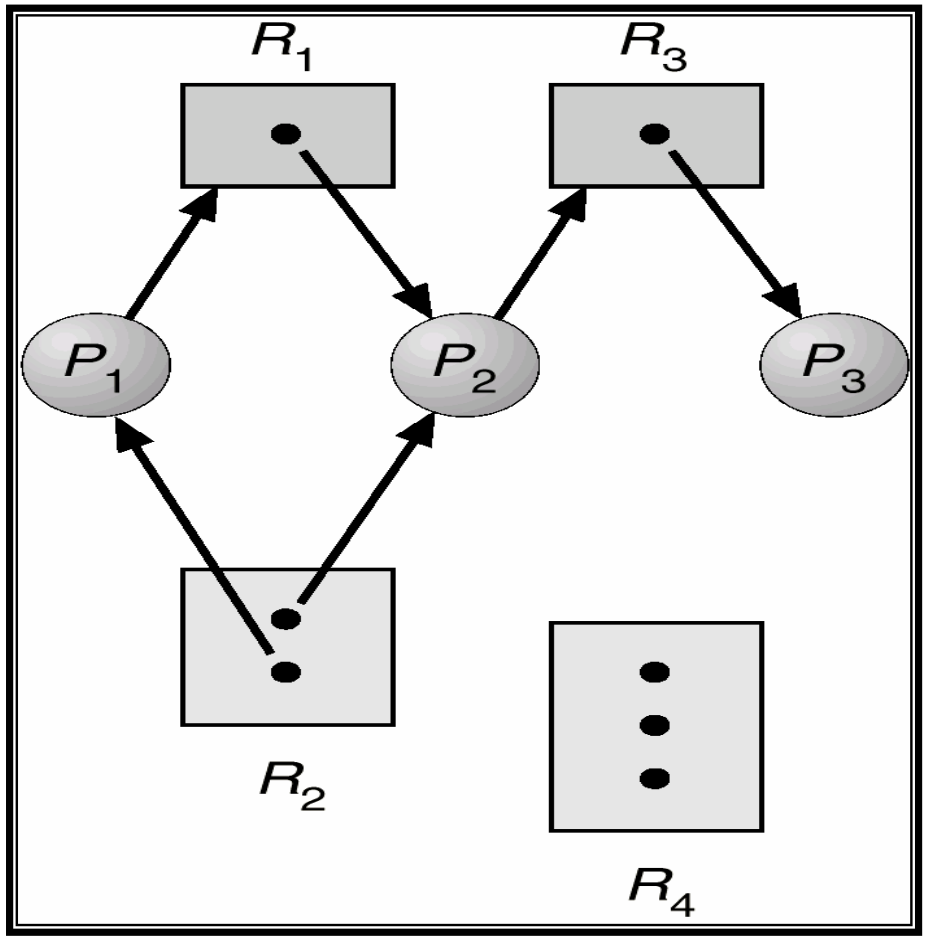
- P_i yêu cầu một thể hiện của R_j



- P_i đang giữ một thể hiện của R_j

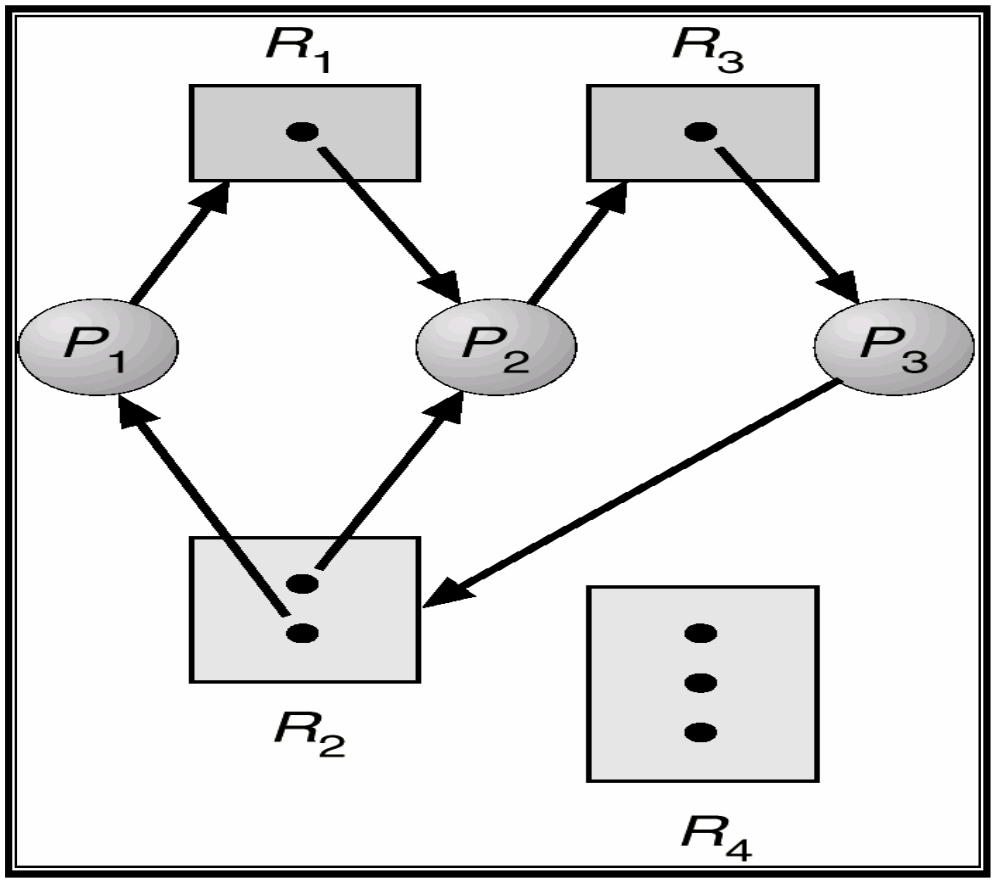


Mô hình hệ thống (4)
Ví dụ đồ thị cấp phát tài nguyên



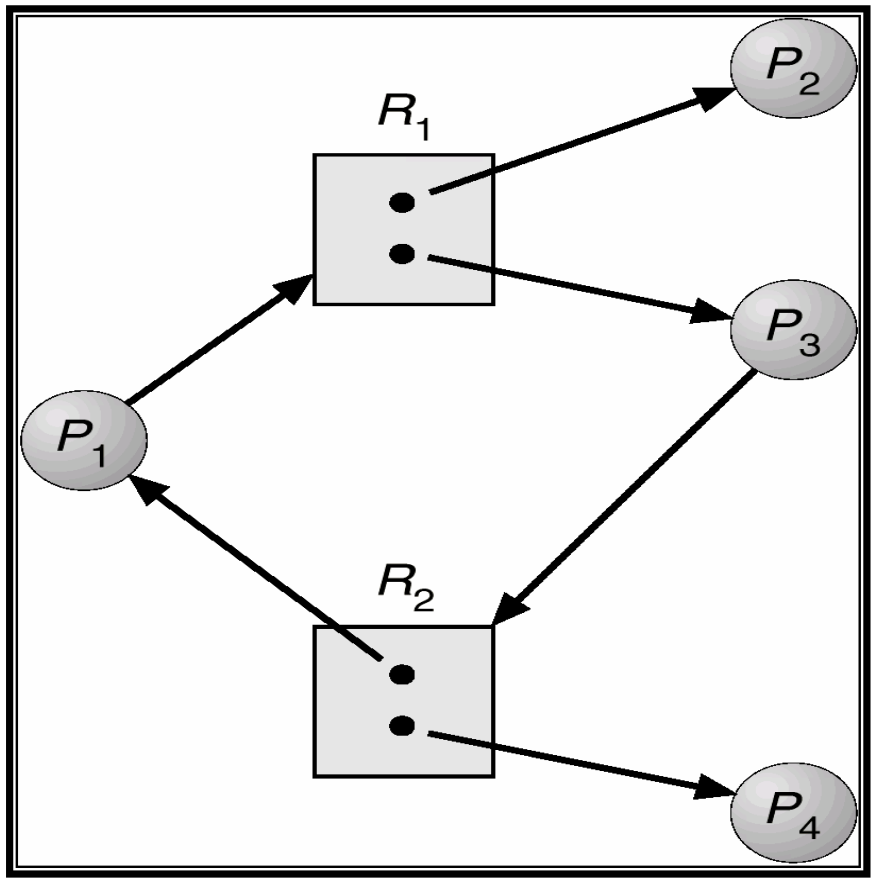
Mô hình hệ thống (5)

Ví dụ đồ thị cấp phát tài nguyên với deadlock



Mô hình hệ thống (6)

Đồ thị cấp phát tài nguyên có chu trình, không deadlock



Tính chất của deadlock (1)
Điều kiện phát sinh deadlock

Deadlock có thể phát sinh nếu 4 điều kiện sau thỏa cùng lúc:

- **Loại trừ hỗ tương:** chỉ một quá trình có thể sử dụng tài nguyên tại một thời điểm.
- **Giữ và chờ:** Một quá trình đang giữ ít nhất là một tài nguyên và đang chờ để đạt được một tài nguyên khác đang bị giữ bởi quá trình khác.
- **Không trưng dụng:** một tài nguyên chỉ có thể được giải phóng một cách tự nguyện bởi quá trình đang giữ nó, sau khi quá trình này hoàn thành.
- **Chờ đợi vòng tròn:** tồn tại một tập hợp $\{P_0, P_1, \ldots, P_n\}$ các quá trình đang chờ đợi như sau: P_0 đang đợi tài nguyên mà P_1 đang giữ, P_1 đang đợi tài nguyên mà P_2 đang giữ, …, P_{n-1} đang đợi tài nguyên mà P_n đang giữ và P_n lại đang chờ đợi tài nguyên mà P_0 đang giữ.

Tính chất của deadlock (2)

Các nhận xét cơ bản về đồ thị cấp phát tài nguyên

- Nếu đồ thị không có chu trình (cycle) ⇨ không có deadlock.
- Nếu đồ thị có một chu trình ⇨
  - Nếu một tài nguyên chỉ có một thể hiện thì deadlock xảy ra.
  - Nếu một tài nguyên có vài thể hiện, có khả năng deadlock xảy ra.

Các biện pháp khống chế deadlock

- Đảm bảo rằng hệ thống sẽ không bao giờ bước vào trạng thái deadlock.
  - Ngăn ngừa dealock.
  - Tránh deadlock.
- Cho phép hệ thống bước vào trạng thái deadlock và sau đó phục hồi lại.
- Bỏ qua vấn đề này và xem như hệ thống sẽ không bao giờ xảy ra deadlock.
  - Biện pháp này được sử dụng trong nhiều hệ điều hành khác nhau, bao gồm cả Unix.

Ngăn chặn deadlock (1)
Giải pháp

Thắt chặt lại các cách thức yêu cầu tài nguyên của quá trình.

- **Loại trừ hỗ tương:** không yêu cầu đối với các tài nguyên có thể chia sẻ; áp dụng đối với các tài nguyên không thể chia sẻ.
- **Giữ và chờ:** phải đảm bảo rằng mỗi khi một quá trình yêu cầu một tài nguyên, nó không đang giữ một tài nguyên khác.
  - Hai giao thức:
    - ✓ Đòi hỏi quá trình yêu cầu và được cấp tất cả các tài nguyên nó cần trước khi bắt đầu thực thi.
    - ✓ Cho phép quá trình yêu cầu tài nguyên chỉ khi nó hiện không giữ một tài nguyên nào cả.
  - Giải pháp này làm giảm đáng kể hiệu xuất sử dụng tài nguyên, và có thể gây ra tình trạng đói tài nguyên.

Ngăn chặn deadlock (2)
Giải pháp

- **Không trưng dụng (no preemption): hai giao thức**
  - Nếu một quá trình đang giữ một số tài nguyên lại yêu cầu thêm một tài nguyên mới, nhưng tài nguyên mới này không thể được cấp phát, thì quá trình đó phải giải phóng tất cả các tài nguyên nó đang giữ.
    - ✓ Các tài nguyên vừa được trưng dụng được thêm vào danh sách các tài nguyên mà quá trình đang cần.
    - ✓ Quá trình sẽ bị khởi động lại chỉ khi nó không thể xin lại được các tài nguyên cũ cũng như tài nguyên mới nó đang cần.
  - Nếu một quá trình yêu cầu tài nguyên, cấp phát chúng nếu chúng sẵn dùng. Nếu không, trưng dụng các tài nguyên được cấp phát cho các quá trình khác, đang chờ đợi được cấp thêm tài nguyên.
- **Chờ đợi vòng tròn (circular wait):** phải áp đặt thứ tự toàn cục của tất cả các lọai tài nguyên và yêu cầu rằng mỗi quá trình phải yêu cầu các tài nguyên theo thứ tự đã cho.

Tránh deadlock (1)
Giải pháp

Yêu cầu thông tin bổ sung về cách tài nguyên được yêu cầu.

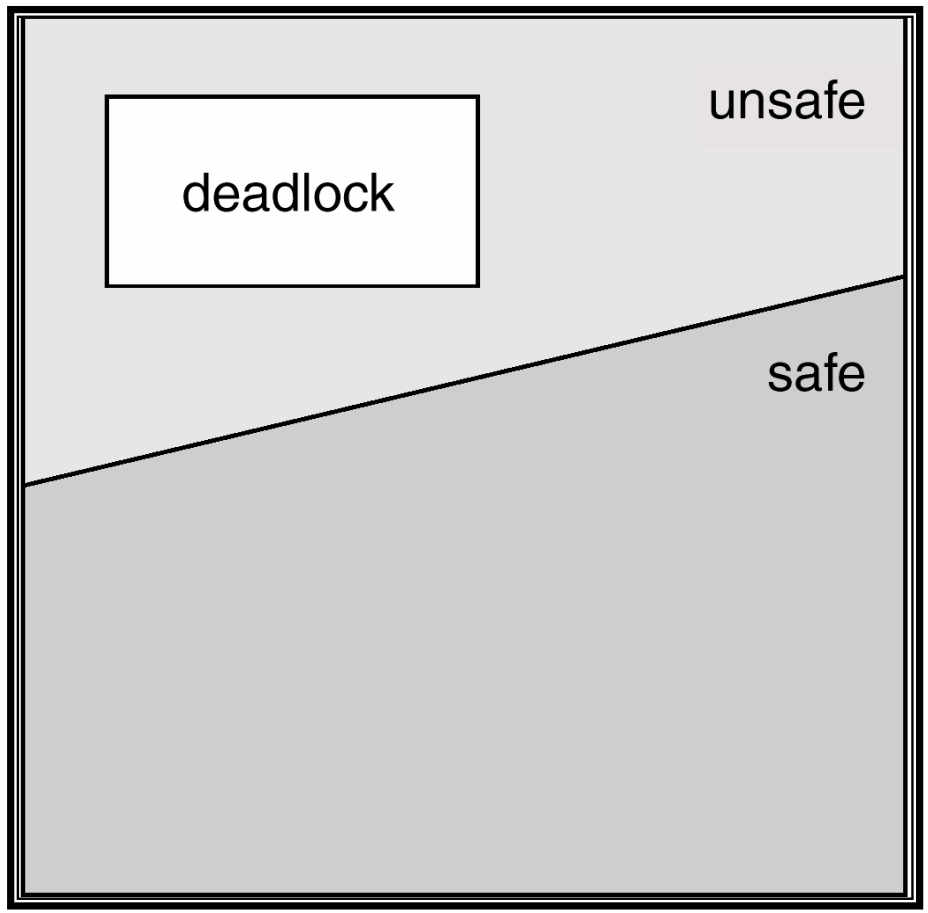
- Mô hình đơn giản và hữu ích nhất là yêu cầu mỗi quá trình khai báo số lượng tối đa của mỗi dạng tài nguyên mà nó cần.
  - Với những thông tin được biết trước, ta có thể xây dựng các giải thuật để bảo đảm rằng hệ thống sẽ không đi vào trạng thái deadlock.
- Giải thuật tránh deadlock sẽ kiểm tra động trạng thái cấp phát tài nguyên để bảo đảm rằng không bao giờ xảy ra chờ đợi vòng tròn.
- Trạng thái cấp phát tài nguyên được định nghĩa bởi số lượng tài nguyên đã được cấp phát và sẵn dùng, và số lượng yêu cầu tối đa của các quá trình.
  - Hai giải thuật sẽ được xem xét:
    - ✓ Giải thuật đồ thị cấp phát tài nguyên (Resource-Allocation-Graph Algorithm).
    - ✓ Giải thuật Banker.

Tránh deadlock (2)
Trạng thái an toàn (Safe State)

- Khi một quá trình yêu cầu một tài nguyên đang sẵn dùng, hệ thống phải quyết định xem việc cấp tài nguyên này tức thời có giữ hệ thống ở trạng thái an toàn hay không.
- Hệ thống ở trạng thái an toàn nếu tồn tại một dãy an toàn (safe sequence) cho tất cả quá trình.
- Dãy $<P_1, P_2, \ldots, P_n>$ được gọi là an toàn nếu với mỗi quá trình P_i, các tài nguyên mà P_i có thể còn yêu cầu có thể được thỏa mãn bởi các tài nguyên đang sẵn dùng + các tài nguyên đang bị giữ bởi tất cả quá trình P_j, với $j<i$.
  - Nếu các nhu cầu tài nguyên của P_i không được làm thõa mãn ngay tức thì, thì P_i có thể đợi đến khi tất cả P_j hoàn thành.
  - Khi P_j hoàn thành, P_i có thể lấy các tài nguyên cần thiết, thực thi tiếp, trả lại số tài nguyên đã chiếm và kết thúc.
  - Khi P_i kết thúc, P_{i+1} có thể lấy các tài nguyên mà nó cần, …

Tránh deadlock (3)
Những nhận xét cơ sở

- Nếu hệ thống ở trong trạng thái an toàn ⇨ không có deadlock.
- Nếu hệ thống ở trong trạng thái không an toàn ⇨ có thể có deadlock.
- Tránh deadlock ⇨ đảm bảo rằng hệ thống sẽ không bao giờ rơi vào trạng thái không an toàn.

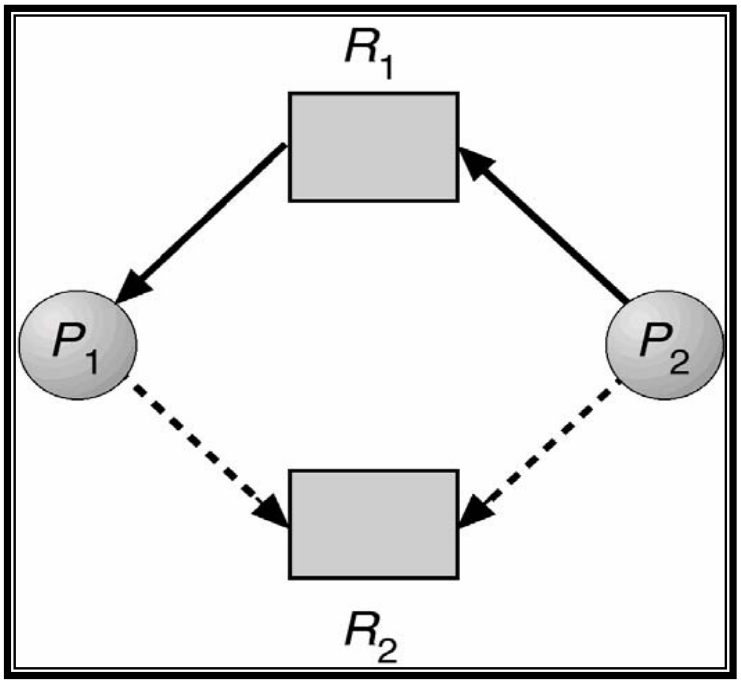


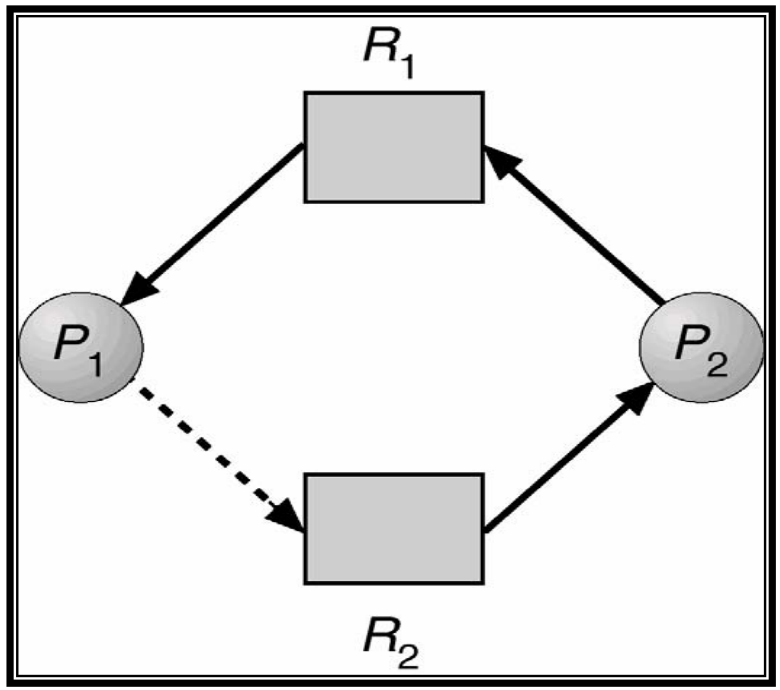
Tránh deadlock (4)
Giải thuật đồ thị cấp phát tài nguyên

- Được áp dụng trong hệ thống chỉ có một thể hiện cho mỗi dạng tài nguyên.
- Cạnh "**dự định yêu cầu**" $P_i \rightarrow R_j$ chỉ ra rằng quá trình P_j có thể yêu cầu tài nguyên R_j; được biểu diễn bởi một đường chấm.
- Cạnh dự định chuyển thành cạnh yêu cầu khi quá trình yêu cầu một tài nguyên.
- Khi một tài nguyên được giải phóng bởi một quá trình, cạnh cấp phát chuyển thành cạnh dự định yêu cầu.
- Tài nguyên phải được dự tính yêu cầu trước trong hệ thống.
  - Các cạnh dự định yêu cầu phải xuất hiện sẵn trong đồ thị.
- Một yêu cầu chỉ được cấp chỉ khi việc chuyển từ $P_i \rightarrow R_j$ sang $R_j \rightarrow P_i$ không tạo ra chu trình trong đồ thị cấp phát tài nguyên.
  - Việc kiểm tra trạng thái an toàn được thực hiện bằng giải thuật phát hiện chu trình (cycle-detection algorithm)

Tránh deadlock (5)

Đồ thị cấp phát tài nguyên dùng tránh deadlock





Yêu cầu tài nguyên của P_2 đối với R_2 sẽ không được cấp phát, mặc dù R_2 đang sẵn dùng, bởi vì có thể tạo ra chu trình, dẫn tới deadlock.

Tránh deadlock (6)
Giải thuật Banker

- Được áp dụng trong hệ thống có nhiều thể hiện cho mỗi dạng tài nguyên.
- Mỗi quá trình phải khai báo số lượng tối đa các thể hiện của mỗi dạng tài nguyên mà nó cần.
- Khi một quá trình yêu cầu một tài nguyên, nó có thể phải chờ đợi.
- Khi một quá trình có được tất cả tài nguyên mà nó cần, nó phải trả lại hết các tài nguyên trong một khoảng thời gian hữu hạn.

Tránh deadlock (7)
Cấu trúc dữ liệu cho giải thuật Banker

Đặt *n* = số lượng quá trình, *m* = số các loại tài nguyên.

- **Available:** vector có chiều dài *m*. Nếu Available [*j*] = *k*, có *k* thể hiện của dạng tài nguyên R_j đang sẵn dùng.
- **Max:** ma trận *n x m*. Nếu Max [*i,j*] = *k*, thì quá trình P_i *có* thể yêu cầu tối đa *k* thể hiện của tài nguyên R_j.
- **Allocation:** *ma trận n* x *m*. Nếu Allocation[*i,j*] = *k* thì P_i đang giữ *k* thể hiện của R_j.
- **Need:** Ma trận *n* x *m*. Nếu *Need*[*i,j*] = *k*, thì P_i có thể cần thêm *k* thể hiện của tài nguyên R_j để có thể hoàn thành công việc của nó.

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

Tránh deadlock (8)
Giải thuật Banker - Giải thuật an toàn

1. Đặt *Work* và *Finish* là các vectors có chiều dài tương ứng là *m* và *n*. Khởi tạo:

   Work = *Available*

   Finish[*i*] = *false* for *i* = 0,1,2,3, …,*n-1.*

2. Tìm *i* để cả hai điều kiện sau thỏa:

   a. *Finish[i]* = *false*

   b. *Need[i]* $\leq$ *Work*

   Nếu không tồn tại *i*, nhảy đến bước 4.

3. *Work* = *Work* + *Allocation[i]*

   Finish[*i*] = *true*

   nhảy đến bước 2.

4. Nếu *Finish* [*i*] == true với mọi *i*, thì hệ thống đang ở trạng thái an toàn.

Tránh deadlock (9)
Giải thuật Banker - Giải thuật yêu cầu tài nguyên

- *Request* = vector yêu cầu cho quá trình P_i. Nếu *Request[i,j]* = *k* thì quá trình P_i muốn *k* thể hiện của tài nguyên R_j.
  1. Nếu *Request[i,j]* $\leq$ *Need[i,j]* nhảy sang bước 2. Ngược lại, báo lỗi do quá trình đã vượt quá số tài nguyên đã dự định sử dụng.
  2. Nếu *Request[i,j]* $\leq$ *Available[j]*, nhảy sang bước 3. Ngược lại P_i phải chờ, do các tài nguyên nó yêu cầu hiện không sẵn dùng.
  3. Giả vờ như đang cấp tài nguyên cho P_i bằng cách sửa đổi trạng thái như sau:

     Available[j] = Available[j] – Request[i,j];

     Allocation[i,j] = Allocation[i,j] + Request[i,j];

     Need[i,j] = Need[i,j] – Request[i,j]

  - Nếu an toàn $\Rightarrow$ tài nguyên được cấp cho P_i.
  - Nếu không an toàn $\Rightarrow$ P_i phải đợi, và trạng thái cấp phát tài nguyên cũ được phục hồi.

Tránh deadlock (10)
Ví dụ của giải thuật Banker

- 5 quá trình từ P_0 đến P_4; 3 loại tài nguyên *A* (10 thể hiện), *B* (5 thể hiện), và *C* (7 thể hiện).
- Hiện trạng tại thời điểm T_0

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Tránh deadlock (11)
Ví dụ của giải thuật Banker

- Nội dung của ma trận Need = Max – Allocation.

	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

- Hệ thống đang ở trạng thái an toàn do dãy < P_1, P_3, P_4, P_2, P_0> thõa mãn tiêu chí về an toàn.

Tránh deadlock (12)
Ví dụ P_1 yêu cầu (1,0,2)

- Kiểm tra $Request_1 \leq Available$ (nghĩa là, $(1,0,2) \leq (3,3,2)) \Rightarrow$ *true.*

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Sự thực hiện giải thuật an toàn cho thấy dãy $<P_1, P_3, P_4, P_0, P_2>$ thõa mãn yêu cầu về an toàn ⇨ yêu cầu của P1 được đáp ứng ngay lập tức.
- Yêu cầu (3,3,0) của P_4 có thể được cấp không?
- Yêu cầu (0,2,0) của P_0 có thể được cấp không?

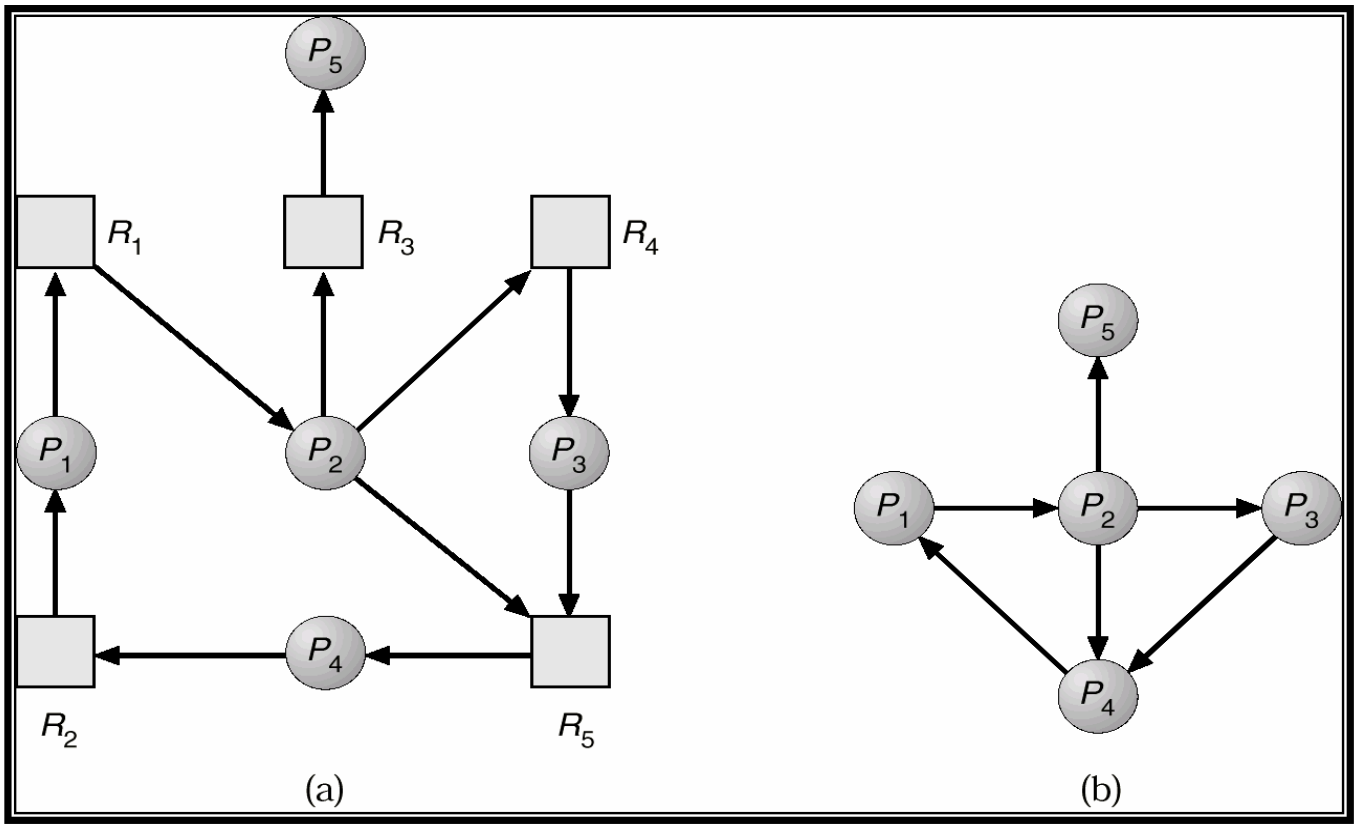
Phát hiện deadlock (1)

- Cho phép hệ thống bước vào trạng thái deadlock.
- Giải thuật phát hiện.
- Sơ đồ phục hồi.

Phát hiện deadlock (2)
Mỗi dạng tài nguyên chỉ có một thể hiện

- Dùng một biến đổi của đồ thị cấp phát tài nguyên, gọi là đồ thị *wait-for*.
- Đồ thị wait-for:
  - Các nút là các quá trình.
  - $P_i \rightarrow P_j$ nếu P_i đang đợi P_j.
- Thường xuyên thực hiện giải thuật tìm kiếm chu trình trong đồ thị.
- Deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị wait-for chứa một chu trình.
- Một giải thuật phát hiện chu trình trong đồ thị yêu cầu theo thứ tự n^2 thao tác, với n là số cạnh trong đồ thị.

Phát hiện deadlock (3)
Đồ thị cấp phát tài nguyên và Wait-for



Đồ thị cấp phát tài nguyên

Đồ thị wait-for tương ứng

Phát hiện deadlock (4)
Mỗi dạng tài nguyên có nhiều thể hiện

- **Available:** một vector có chiều dài *m* chỉ ra số lượng thể hiện còn sẵn dùng của mỗi loại tài nguyên.
- **Allocation:** một ma trận *n* x *m* định nghĩa số lượng thể hiện của mỗi loại tài nguyên hiện đang được cấp phát cho mỗi quá trình.
- **Request:** một ma trận *n* x *m* chỉ ra lượng yêu cầu của mỗi quá trình. Nếu *Request* [*i,j*] = *k*, thì quá trình P_i đang yêu cầu thêm *k* thể hiện của tài nguyên loại R_j.

Phát hiện deadlock (5)
Giải thuật phát hiện deadlock

1. Đặt *Work* và *Finish* là các vectors có chiều dài tương ứng *m* và *n*, khởi tạo:
   a. *Work* = *Available*
   b. For *i* = 0,1,2, …, *n*-1
      if *Allocation[i]* $\neq$ 0 then *Finish[i]* = false;
      else *Finish[i]* = *true;*
2. Tìm ra chỉ số *i* để hai điều kiện sau được thỏa:
   a. *Finish[i]* == *false*
   b. *Request[i]* $\leq$ *Work*

   Nếu không tồn tại *i*, nhảy đến bước 4.
3. *Work* = *Work* + *Allocation[i]*
   Finish[*i*] = *true*
   Nhảy đến bước 2.
4. If *Finish*[*i*] == *false* cho vài *i*, $1 \leq i \leq n$, then hệ thống đang ở trạng thái deadlock. Ngoài ra, if *Finish*[*i*] == *false*, then P_i đang bị deadlock.

Giải thuật yêu cầu O(m x $n^{2)}$ thao tác.

Phát hiện deadlock (6)
Ví dụ về giải thuật phát hiện deadlock

- Năm quá trình P_0 đến P_4; có 3 kiểu tài nguyên A (7 thể hiện), *B* (2 thể hiện), và *C* (6 thể hiện).
- Hiện trạng tại thời điểm T_0:

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Dãy <P_0, P_2, P_3, P_1, P_4> sẽ dẫn đến *Finish*[*i*] = *true* với mọi *i*. Hệ thống không trong trạng thái deadlock.

Phát hiện deadlock (7)
Ví dụ về giải thuật phát hiện deadlock

- P_2 yêu cầu thêm một thể hiện của tài nguyên loại *C*.

	Request
	A B C
P_0	0 0 0
P_1	2 0 1
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- Trạng thái của hệ thống?
  - P_0 không yêu cầu thêm tài nguyên nào, nhưng hệ thống vẫn không đủ tài nguyên để thỏa nhu cầu của các quá trình khác.
  - Deadlock xảy ra, bao gồm các quá trình P_1, P_2, P_3, và P_4.

Phát hiện deadlock (8)
Việc sử dụng giải thuật phát hiện deadlock

- Sử dụng giải thuật khi nào và thường xuyên như thế nào sẽ phụ thuộc vào:
  - Việc deadlock xảy ra thường xuyên như thế nào?
  - Bao nhiêu quá trình bị ảnh hưởng bởi deadlock, cần phải quay lại (rollback) khi nó xuất hiện?
    - ✓ Cần một quá trình để mở một chu trình (cycle)
- Nếu giải thuật phát hiện deadlock xảy ra quá ít, thì có thể sẽ có nhiều chu trình xuất hiện trong đồ thị. Khi đó, khó có thể biết rằng quá trình nào trong số các quá trình bị deadlock đã gây ra deadlock.

Phục hồi từ Deadlock (1)
Giải pháp

- Khi phát hiện deadlock, một số cách có thể dùng để phục hồi từ deadlock:
  - Phục hồi bằng tay: cho phép thao tác viên phục hồi bằng tay.
  - Phục hồi tự động.
- Hai tùy chọn có thể dùng để xóa deadlock:
  - Ngưng một hoặc nhiều quá trình để xóa các chu trình sinh deadlock
  - Trưng dụng một hoặc nhiều tài nguyên từ một hoặc nhiều quá trình bị deadlock.

Phục hồi từ Deadlock (2)
Ngưng quá trình (Process Termination)

- Hai phương pháp:
  - Hủy bỏ tất cả các quá trình bị deadlock
    - ✓ Chi phí lớn: do quá trình có thể đã tính toán trong thời gian dài ⇨ việc tính toán lại sẽ mất nhiều thời gian.
  - Hủy bỏ mỗi lần một quá trình đến khi chu trình deadlock bị loại trừ.
    - ✓ Chi phí cũng phải được xem xét, vì sau mỗi bước phải chạy lại giải thuật phát hiện deadlock.
- Thế chúng ta nên hủy bỏ các quá trình theo thứ tự nào?
  - Độ ưu tiên của quá trình.
  - Quá trình đã diễn ra lâu chưa và nó còn tiếp diễn bao lâu nữa.
  - Số tài nguyên mà quá trình đã sử dụng.
  - Số tài nguyên mà quá trình cần để hoàn thành.
  - Có bao nhiêu quá trình cần phải được kết thúc.
  - Quá trình là tương tác hay theo lô.

Phục hồi từ deadlock
Trưng dụng tài nguyên (Resource Preemption)

- Chọn ra một nạn nhân:
  - Chọn tài nguyên và quá trình nào bị trưng dụng.
  - Xác định thứ tự trưng dụng để tối thiểu hóa chi phí.
- Quay lại (rollback):
  - Đưa quá trình quay lai một trạng thái an toàn nào đó.
  - Khởi động lại quá trình từ trạng thái đó.
  - Đòi hỏi hệ thống phải lưu lại thông tin về trạng thái an toàn của tất cả các quá trình đang chạy.
- Đói tài nguyên (starvation):
  - Tránh tình trạng một quá trình có thể liên tục bị chọn là nạn nhân.

Giải pháp tích hợp để khống chế deadlock

- Kết hợp 3 giải pháp cơ bản: ngăn chặn, tránh, phát hiện.
- Phân chia các tài nguyên thành các dạng theo thứ tự phân cấp.
  - Cho phép áp dụng một giải pháp tối ưu cho quản lý deadlock cho mỗi loại tài nguyên trong hệ thống.
- Ví dụ:
  - **Tài nguyên bên trong (internal resource)**: tài nguyên được dùng bởi hệ thống, như process control block. Dùng phương pháp ngăn ngừa thông qua việc sắp xếp thứ tự tài nguyên.
  - **Bộ nhớ trung tâm (central memory)**: bộ nhớ cho các quá trình người dùng. Dùng phương pháp ngăn ngừa thông qua trưng dụng.
  - **Tài nguyên công việc (job resource)**: thiết bị có thể gán (như tape drive) và các tập tin. Dùng phương pháp tránh deadlock.
  - **Không gian có thể hoán chuyển (swappable space)**: không gian cho mỗi quá trình cho việc hoán chuyển ra dĩa. Dùng phương pháp cấp phát trước (preallocation).