Chương 5
BẾ TẮC

Bế tắc (Deadlock)

- Vấn đề “Bế tắc”
- Mô hình hệ thống
- Các đặc điểm của bế tắc
- Các phương pháp xử lý bế tắc
- Ngăn chặn bế tắc
- Tránh bế tắc
- Khôi phục sau bế tắc

Vấn đề "Bế tắc"

- Một tập các tiến trình bị chặn, mỗi tiến trình giữ một tài nguyên và chờ một tài nguyên bị chiếm giữ bởi một tiến trình khác trong tập
- Ví dụ
  - Một hệ thống có 2 băng từ
  - P1 và P2 mỗi tiến trình giữ một băng từ và đòi hỏi băng từ được giữ bởi tiến trình kia.
- Ví dụ
  - semaphores A và B, được khởi tạo bằng 1

P0	P1
wait(A);	wait (B)
wait(B);	wait (A)

Mô hình hệ thống

- Các kiểu tài nguyên $R_1, R_2, \ldots, R_n$
  - CPU cycles, memory space, I/O devices
- Mỗi tài nguyên R_i có W_i thể hiện.
- Mỗi tiến trình sử dụng tài nguyên như sau:
  - request
  - use
  - release

Các đặc điểm của bế tắc

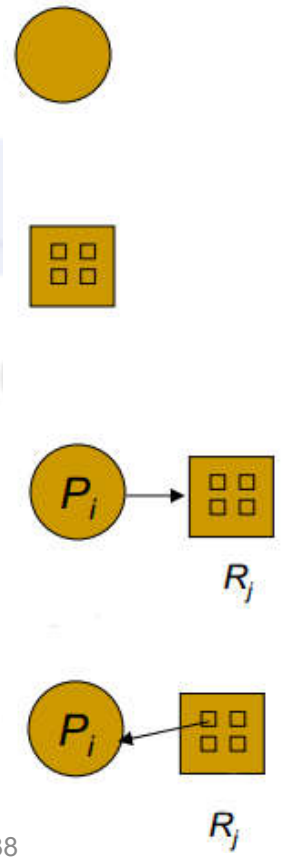
- Bế tắc có thể xảy ra nếu cả bốn điều kiện sau xảy ra đồng thời
  - **Độc quyền truy xuất (Mutual Exclusion):** chỉ một tiến trình được sử dụng tài nguyên tại một thời điểm.
  - **Giữ và chờ (Hold and wait):** một tiến trình giữ ít nhất một tài nguyên và chờ các tài nguyên khác đang được giữ bởi các tiến trình khác.
  - **Không chiếm đoạt (No preemption):** một tài nguyên chỉ có thể được giải phóng một cách tự nguyện bởi tiến trình giữ nó, sau khi tiến trình đó hoàn thành công việc của nó.
  - **Chờ đợi vòng tròn (Circular wait):** Một tập các tiến trình đang chờ $\{P_0, P_1, P_2, \dots, P_n\}$, trong đó P_0 chờ một tài nguyên bị giữ bởi P_1, P_1 chờ một tài nguyên bị giữ bởi $P_2, \dots, P_{n-1}$ chờ một tài nguyên được giữ bởi P_n, và P_n đang chờ một tài nguyên được giữ bởi P_0

Đồ thị phân phối tài nguyên

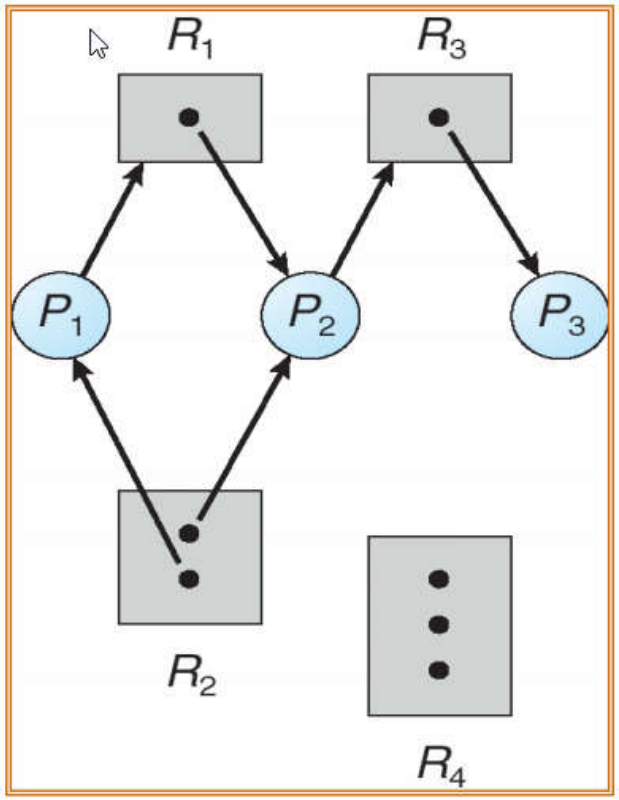
- Có một tập các đỉnh V và một tập các cạnh E
  - V được chia thành hai tập:
    - $P = \{P_1, P_2, \dots, P_n\}$, tập chứa tất cả các tiến trình trong hệ thống.
    - $R = \{R_1, R_2, \dots, R_m\}$, tập chứa tất cả các kiểu tài nguyên trong hệ thống.
  - Cạnh yêu cầu (request) – cạnh có hướng $P_i \rightarrow R_j$
  - Cạnh phân phối (assignment) – cạnh có hướng $R_j \rightarrow P_i$

Đồ thị phân phối tài nguyên

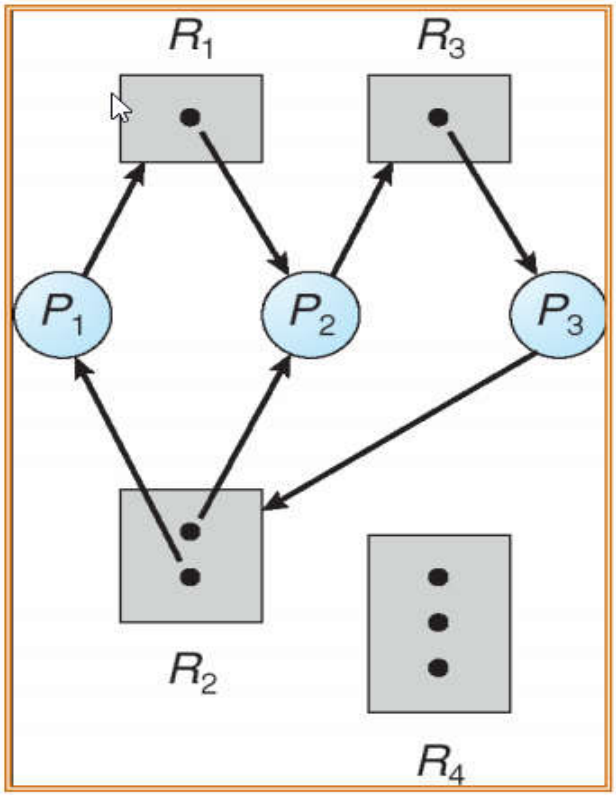
- Tiến trình
- Kiểu tài nguyên với 4 thể hiện
- P_i yêu cầu một thể hiện của R_j
- P_i giữ một thể hiện của R_j



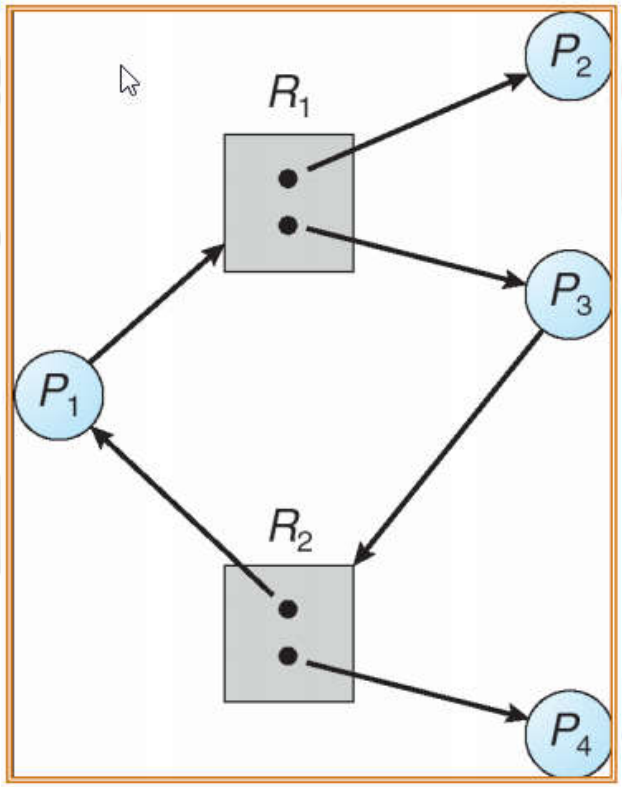
Ví dụ về một đồ thị phân phối tài nguyên



Đồ thị phân phối tài nguyên với một bế tắc



Đồ thị với một chu trình nhưng không có bế tắc



Tiên đề

- Nếu một đồ thị không có chu trình ⇒ không có bế tắc.
- Nếu đồ thị có một chu trình ⇒
  - Nếu mỗi tài nguyên chỉ có một thể hiện thì xuất hiện bế tắc.
  - Nếu mỗi loại tài nguyên có một vài thể hiện thì có thể có bế tắc

Các phương pháp xử lý bế tắc

- Đảm bảo hệ thống sẽ không bao giờ đi vào trạng thái bế tắc
- Cho phép hệ thống vào trạng thái bế tắc sau đó khôi phục
- Bỏ qua vấn đề này và xem rằng bế tắc không bao giờ xảy ra trong hệ thống; được sử dụng bởi hầu hết các hệ điều hành, bao gồm cả UNIX.

Ngăn chặn bế tắc...

- Ràng buộc cách thức yêu cầu tài nguyên của các tiến trình
  - **Độc quyền truy xuất (Loại trừ lẫn nhau)** – không cần thỏa mãn với các tài nguyên có thể chia sẻ; cần thỏa mãn với các tài nguyên không thể chia sẻ (ví dụ máy in).
  - **Giữ và chờ** – phải đảm bảo rằng khi một tiến trình yêu cầu một tài nguyên, nó không được giữ bất kì tài nguyên nào khác.
    - Tiến trình phải yêu cầu và được phân phối tất cả các tài nguyên trước khi nó bắt đầu thực thi
    - Cho phép tiến trình yêu cầu tài nguyên chỉ khi nó không chiếm hữu tài nguyên nào.
    - Tính tận dụng tài nguyên thấp; có thể xảy ra “chết đói”.

...Ngăn chặn bế tắc

- **Không chiếm đoạt**
  - Nếu một tiến trình hiện đang giữ một số tài nguyên nào đó và yêu cầu tài nguyên khác (chưa thể phân phối cho nó ngay lập tức), tất cả các tài nguyên đang bị giữ sẽ được giải phóng.
  - Các tài nguyên bị chiếm đoạt được thêm vào danh sách các tài nguyên mà tiến trình đó đang chờ.
  - Tiến trình sẽ bắt đầu thực thi lại chỉ khi nó có thể lấy lại các tài nguyên cũ cũng như chiếm cả tài nguyên mới mà nó đang yêu cầu.
- **Chờ đợi vòng tròn** – áp đặt một trình tự tổng thể của tất cả các loại tài nguyên, và ràng buộc các tiến trình yêu cầu tài nguyên theo thứ tự tăng dần.

Tránh bế tắc

- Yêu cầu hệ thống phải có trước một số thông tin (prior information available).
  - Mô hình đơn giản: mỗi tiến trình khai báo số tài nguyên lớn nhất thuộc mỗi loại mà nó cần.
  - Thuật toán tránh bế tắc kiểm tra trạng thái phân phối tài nguyên để đảm bảo rằng không bao giờ có điều kiện “chờ đợi vòng tròn” xảy ra.
  - Trạng thái phân phối tài nguyên được xác định bằng số các tài nguyên rỗi, số tài nguyên đã được phân phối và số cực đại yêu cầu của các tiến trình

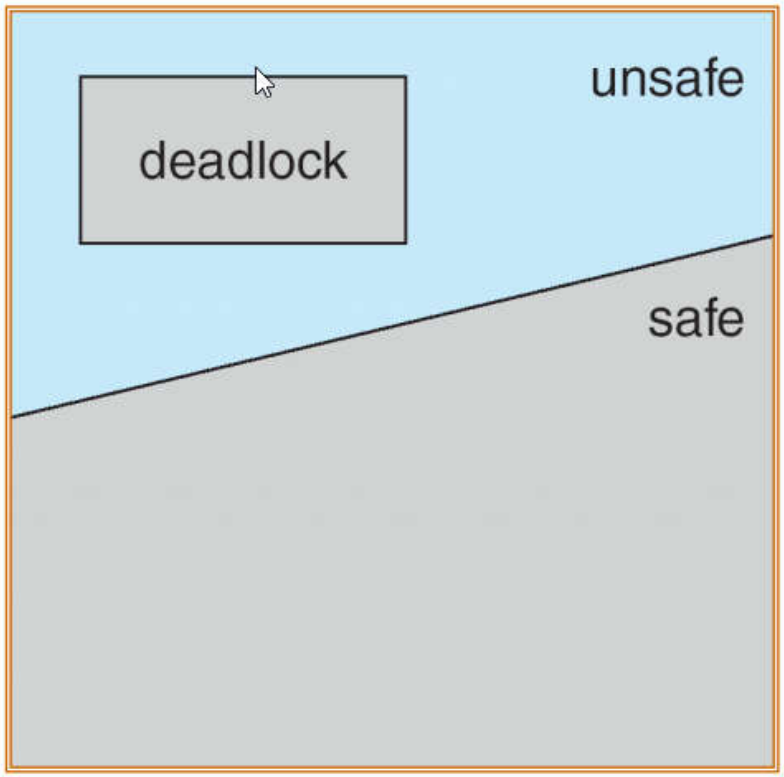
Trạng thái an toàn

- Khi nào một tiến trình yêu cầu một tài nguyên rỗi, cần xác định xem việc phân phối đó có đặt hệ thống vào trạng thái không an toàn hay không.
- Hệ thống trong trạng thái an toàn nếu tồn tại một "chuỗi an toàn" của tất cả các tiến trình.
- Chuỗi $< P_1, P_2, \ldots, P_n >$ là an toàn đối với mỗi P_i, nếu các tài nguyên mà P_i cần đều có thể phân phối bởi các tài nguyên đang rỗi hoặc các tài nguyên đang bị chiếm hữu bởi tất cả các tiến trình P_j (với j<i).
  - Nếu P_i cần tài nguyên không rỗi ngay lập tức, thì P_i có thể đợi cho đến khi tất cả các P_j hoàn thành.
  - Khi P_j đã hoàn thành, P_i có thể chiếm hữu tài nguyên mà nó cần, thực thi và trả lại tài nguyên đã được phân phối và kết thúc.
  - Khi P_i kết thúc, P_{i+1} có thể chiếm hữu tài nguyên mà nó cần

Tiên đề

- Nếu một hệ thống trong trạng thái an toàn ⇒ không có bế tắc
- Nếu một hệ thống ở trong trạng thái không an toàn ⇒ có thể có bế tắc.
- Tránh bế tắc ⇒ đảm bảo rằng hệ thống không bao giờ rơi vào trạng thái không an toàn.

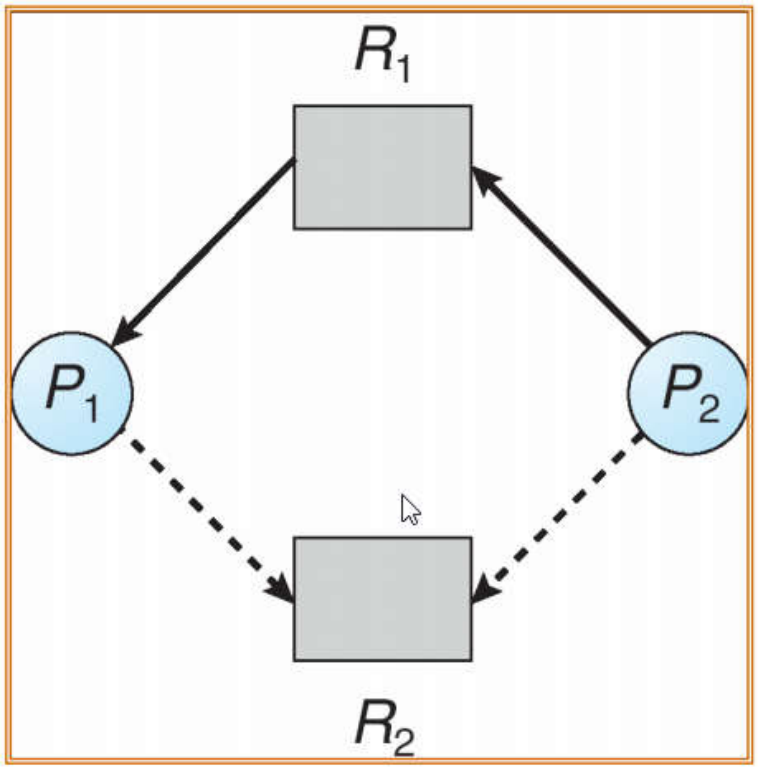
Trạng thái an toàn, không an toàn và bế tắc



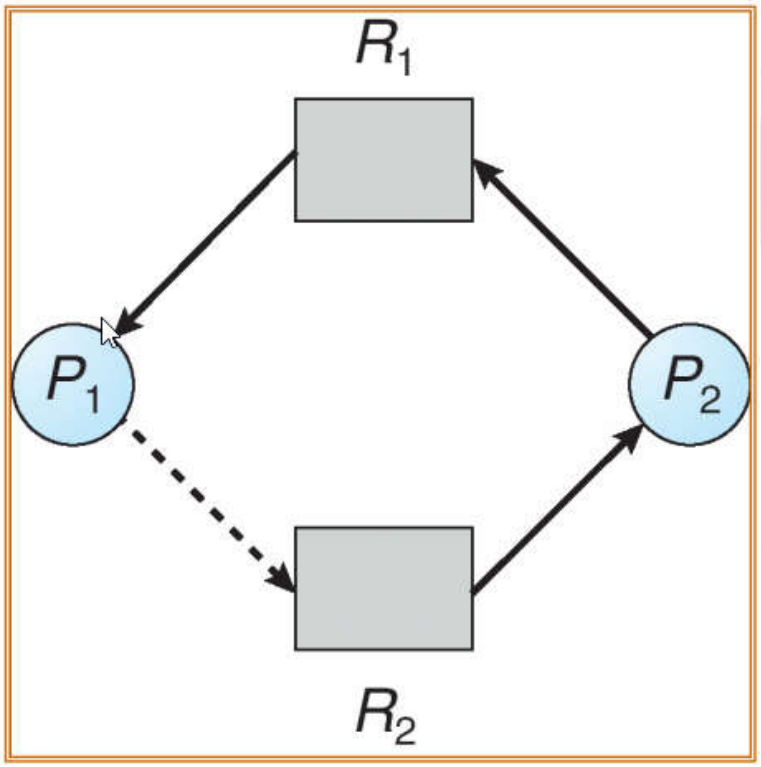
Thuật toán đồ thị phân phối tài nguyên

- Claim $P_i \rightarrow R_j$ chỉ rằng một tiến trình P_i có thể yêu cầu tài nguyên R_j; được biểu diễn bởi đường nét đứt.
- Cạnh Claim được biến đổi thành cạnh request nếu một tiến trình yêu cầu một tài nguyên.
- Khi một tiến trình giải phóng tài nguyên, cạnh assignment được chuyển lại thành cạnh Claim.
- Các tài nguyên phải có một độ ưu tiên trong hệ thống.

Đồ thị phân phối tài nguyên để tránh bế tắc



Trạng thái không an toàn trong đồ thị phân phối tài nguyên



Thuật toán ngân hàng (Banker)

- Các tài nguyên có nhiều thể hiện.
- Mỗi tiến trình khi mới vào hệ thống cần phải khai báo số cực đại tài nguyên mà nó cần.
- Khi một tiến trình yêu cầu tài nguyên, hệ thống kiểm tra xem liệu việc phân phối tài nguyên đó có đảm bảo hệ thống trong trạng thái an toàn hay không
  - Nếu không, tiến trình có thể phải chờ
- Khi một tiến trình đã được phân phối tài nguyên, nó phải trả lại các tài nguyên này trong một thời gian xác định

Cấu trúc dữ liệu cho thuật toán Banker

- Gọi n = số tiến trình, và m = số loại tài nguyên.
- Available: vector độ dài m. Nếu Available[j] = k, có k thể hiện của tài nguyên R_j rỗi.
- Max: ma trận cỡ n x m. Nếu Max[i,j] = k, tiến trình P_i có thể đòi hỏi nhiều nhất là k thể hiện của loại tài nguyên R_j
- Allocation: ma trận cỡ n x m. Nếu Allocation[i,j] = k, tiến trình P_i hiện được phân phối k thể hiện của R_j.
- Need: ma trận cỡ n x m. Nếu Need[i,j] = k, tiến trình P_i có thể cần thêm k thể hiện của R_j để hoàn thành nhiệm vụ của nó.
- Need[i,j] = Max[i,j] – Allocation[i,j].

Thuật toán Safety

1. Gọi Work and Finish là các vector có độ dài lần lượt là m và n. Khởi tạo:
   - Work = Available
   - Finish [i] =false for i =0,1, …, n-1.
2. Tìm i thỏa mãn hai điều kiện sau:
   - (a) Finish [i] = false
   - (b) $Need_i \leq Work$
   - Nếu không tìm được i, nhảy đến bước 4.
3. $Work = Work + Allocation_i$
   - Finish [i] = true
   - Nhảy đến bước 2.
4. Nếu Finish[i] == true với tất cả i, hệ thống trong trạng thái an toàn

Thuật toán phân phối yêu cầu tài nguyên cho tiến trình P_i

$Request_i$ = vector yêu cầu của P_i. Nếu $Request_i[j] = k$, P_i cần k thể hiện của R_j.

1. Nếu $Request_i \leq Need_i$ nhảy đến bước 2. Nếu không, lỗi (tiến trình cần nhiều hơn số yêu cầu khai báo ban đầu).

2. Nếu $Request_i \leq Available$, nhảy đến bước 3. Nếu không P_i phải chờ vì tất cả tài nguyên đều không rỗi.

3. Thử phân phối tài nguyên được yêu cầu cho P_i bằng cách thay đổi trạng thái như sau:

$Available = Available - Request_i$;

$Allocation_i = Allocation_i + Request_i$;

$Need_i = Need_i - Request_i$;

- Nếu an toàn ⇒ tài nguyên được phân phối cho P_i.
- Nếu không an toàn ⇒ P_i phải đợi, trạng thái phân phối tài nguyên cũ được khôi phục lại

Ví dụ về thuật toán Banker...

- 5 tiến trình từ P_0 đến P_4
- 3 loại tài nguyên A (10 thể hiện), B (5 thể hiện), và C (7 thể hiện).
- Hệ thống 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	

...Ví dụ

- Nội dung của ma trận Need. Need được định nghĩa là 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 trong trạng thái an toàn vì chuỗi < P_1, P_3, P_4, P_2, P_0> thỏa mãn điều kiện an toàn

Ví dụ: P_1 yêu cầu (1,0,2)...

- Kiểm tra thấy Request ≤Available (hay, (1,0,2) ≤ (3,3,2) ⇒ true).

	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 1	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

- Thực thi thuật toán safety cho thấy chuỗi <P_1, P_3, P_4, P_0, P_2> thỏa mãn yêu cầu an toàn.
- Yêu cầu (3,3,0) của P_4 có thể được gán hay không?
- Yêu cầu (0,2,0) của P_0 có thể được gán hay không?

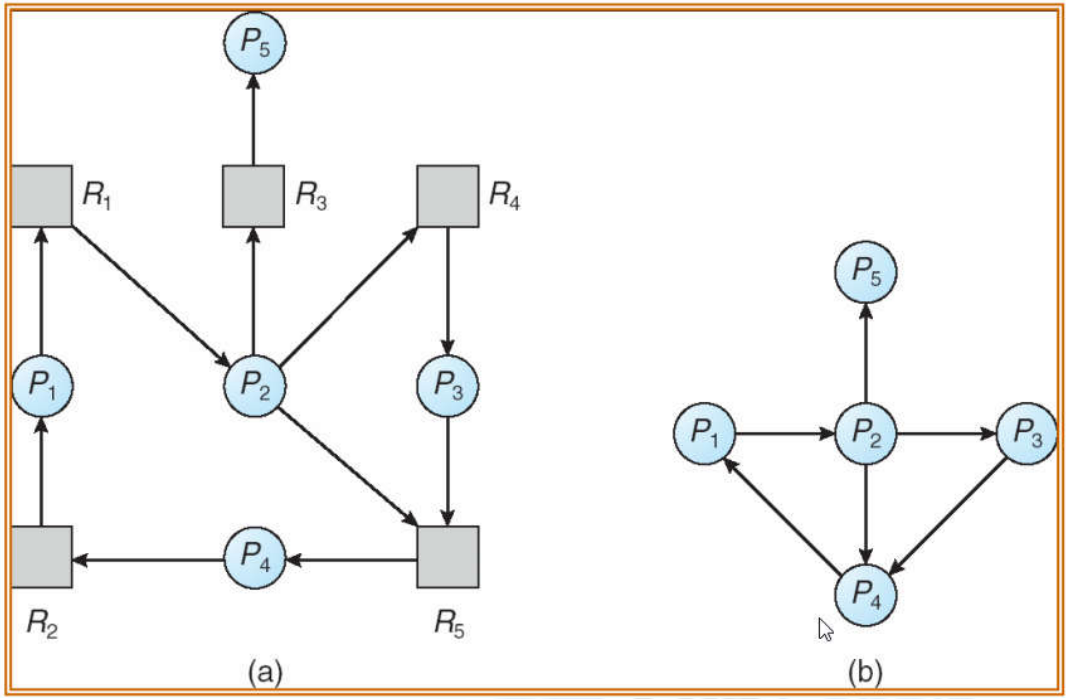
Khôi phục sau bế tắc

- Cho phép vào trạng thái bế tắc
- Thuật toán phát hiện bế tắc
- Phương pháp khôi phục

Mỗi loại tài nguyên có một thể hiện

- Duy trì đồ thị wait-for
  - Các nút là các tiến trình.
  - $P_i \rightarrow P_j$ nếu P_i đang chờ P_j.
- Định kì thực hiện thuật toán tìm chu trình trong đồ thị.
- Một giải thuật kiểm tra chu trình trong đồ thị cần n^2 thao tác, ở đây n là số đỉnh trong đồ thị

Đồ thị phân phối tài nguyên và đồ thị wait-for



(a) Đồ thị phân phối tài nguyên (b) Đồ thị wait-for tương ứng

Mỗi tài nguyên có nhiều hơn một thể hiện

- Available:Vector độ dài m chỉ số các trường hợp còn rỗi đối với mỗi tài nguyên.
- Allocation: Ma trận $n \times m$ xác định số tài nguyên của mỗi loại được phân phối cho tiến trình.
- Request: Ma trận $n \times m$ chỉ yêu cầu hiện tại của tiến trình. Nếu Request $[i, j]$ = k, tiến trình P_i đang cần thêm k thể hiện tài nguyên R_j.

Thuật toán phát hiện bế tắc

1. Gọi Work and Finish là các vector độ dài m và n tương ứng. Khởi tạo:

    (a) Work = Available

    (b)For i = 1,2, …,n, if $Allocation_i \neq 0$, then

        Finish[i] = false; otherwise, Finish[i] = true.

2. Tìm một chỉ số i thỏa mãn:

    (a)Finish[i] == false

    (b)$Request_i$ ≤Work

    Nếu không tồn tại i nhảy đến bước 4.

3.Work= Work+ $Allocation_i$

    Finish[i] = true

    go to step 2.

4.If Finish[i] == false, for some i, 1 ≤i ≤ n, then hệ thống trong trạng thái bế tắc. Hơn nữa, if Finish[i] == false, then P_i bị bế tắc.

Độ phức tạp của thuật toán là O($m \times n^2$).

Ví dụ về thuật toán phát hiện bế tắc

- 5 tiến trình P_0 đến P_4; ba loại tài nguyên: A (7 thể hiện), B (2 thể hiện), and C (6 thể hiện).
- Hệ thống tại thời điểm T_0:

	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

- Chuỗi <P_0, P_2, P_3, P_1, P_4> dẫn đến Finish[i] = true với mọi i.

...Ví dụ về thuật toán phát hiện bế tắc

- P_2 yêu cầu thêm một thể hiện của 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?
  - Có thể lấy lại tài nguyên do tiến trình P_0 nắm giữ, nhưng không đủ tài nguyên để thực hiện các tiến trình khác; các yêu cầu.
  - Tồn tại bế tắc, bao gồm các tiến trình P_1, P_2, P_3, và P_4.

Sử dụng thuật toán phát hiện bế tắc

- Thời điểm, mức độ thường xuyên phụ thuộc vào:
  - Mức độ thường xuyên của bế tắc?
  - Bao nhiêu tiến trình cần phải quay lui?
    - Một đối với mỗi chu trình (các chu trình không giao)
- Nếu thuật toán phát hiện được gọi tùy ý, có thể có rất nhiều chu trình trong đồ thị tài nguyên và vì thế chúng ta không thể phát hiện tiến trình nào trong số các tiến trình đó gây ra bế tắc.

Khôi phục sau bế tắc: Kết thúc tiến trình

- Dừng thực thi của toàn bộ tiến trình bế tắc.
- Dừng từng tiến trình một cho đến khi mất đi chu trình bế tắc.
- Thứ tự dừng?
  - Độ ưu tiên của tiến trình.
  - Tiến trình đã được thực thi và còn phải thực thi trong bao lâu.
  - Các tài nguyên mà tiến trình hiện đang sử dụng.
  - Các tài nguyên mà tiến trình cần thêm.
  - Số tiến trình cần phải kết thúc.
  - Tiến trình là tương tác hay lô?

Khôi phục sau bế tắc: chiếm đoạt tài nguyên

- Lựa chọn một nạn nhân – cực tiểu hóa chi phí.
- Rollback – trở về trạng thái an toàn, khởi động lại tiến trình cho trạng thái đó.
- Chết đói – một tiến trình có thể luôn bị lựa chọn là nạn nhân

38


Question