CT107. HỆ ĐIỀU HÀNH
CHƯƠNG 6. DEADLOCK (KHÓA CHẾT)

Giảng viên: PGS. TS. Trần Cao Đệ (tcde@ctu.edu.vn)

Bộ môn Công Nghệ Thông Tin - Khoa Công Nghệ

Thông Tin & Truyền Thông – Trường Đại học Cần Thơ

2020

MỤC TIÊU

- Mô tả tình trạng deadlock của hệ thống - một trạng thái mà các tiến trình không thể tiến triển để hoàn thành các tác vụ của chúng.
- Trình bày các phương pháp để ngăn chặn hoặc tránh deadlock; và các biện pháp để phát hiện và phục hồi hệ thống một khi deadlock xảy ra.

NỘI DUNG

GIỚI THIỆU DEADLOCK

Deadlock là gì?

Điều kiện phát sinh deadlock Mô hình hóa hệ thống

Đồ thị cấp phát tài nguyên (Resourece Allocation Graph - RAG)

CÁC CÁCH TIẾP CẬN VỚI VẤN ĐỀ DEADLOCK

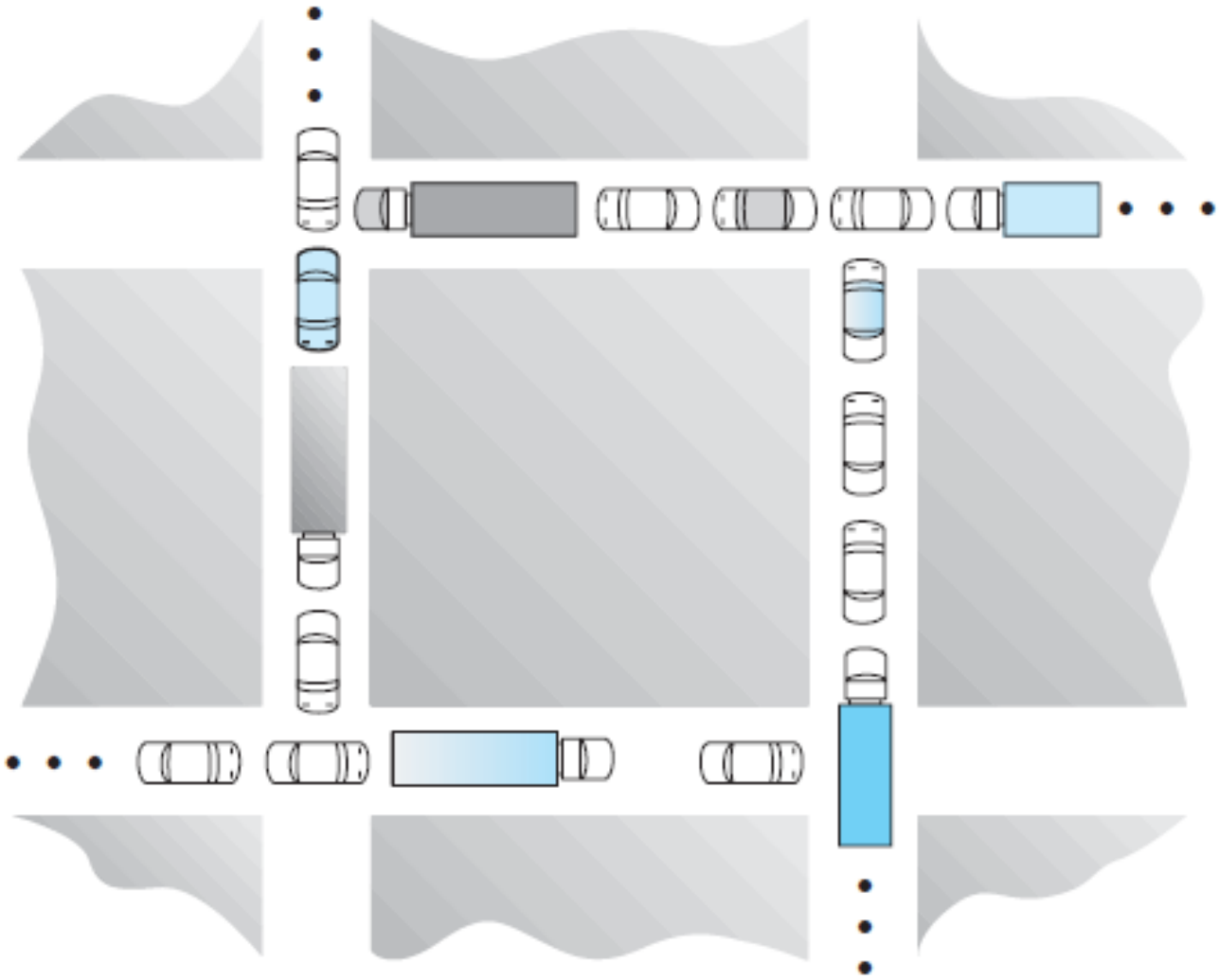
NGĂN CHẶN VÀ TRÁNH DEADLOCK

PHÁT HIỆN VÀ PHỤC HỒI DEADLOCK

DEADLOCK LÀ GÌ?

- **Deadlock** là một trạng thái của hệ thống trong đó:
  - một tập hợp các tiến trình đang bị nghẽn
  - mỗi tiến trình đang giữ một tài nguyên và cũng đang chờ một tài nguyên đang bị giữ bởi một tiến trình khác trong tập các tiến trình đang bị nghẽn.
- Ví dụ 1:
  - Giả sử 1 hệ thống có 2 tiến trình P và Q và F1, F2 là 2 tập tin.
  - Tiến trình P đang giữ F1 và cần truy xuất thêm F2.
  - Tiến trình Q đang giữ F2 và cần truy xuất thêm F1.

VÍ DỤ 2 - TRAFFIC DEADLOCK



ĐIỀU KIỆN PHÁT SINH DEADLOCK

1. **Loại trừ hỗ tương**: ít nhất một tài nguyên được giữ ở chế độ không chia sẻ (nonsharable mode).
2. **Giữ và chờ**: một tiến trình đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên đang bị giữ bởi tiến trình khác.
3. **Không trưng dụng tài nguyên**: không trưng dụng tài nguyên cấp phát tiến trình, trừ khi tiến trình tự hoàn trả.
4. **Chờ đợi vòng tròn**: tồn tại một tập các tiến trình $\{P_0, P_1, ..., Pn\}$ đang chờ đợi như sau: P_0 đợi một tài nguyên P_1 đang giữ, P_1 đợi một tài nguyên P_2 đang giữ, . . . , P_n đang đợi một tài nguyên P_0 đang giữ.

GIỚI THIỆU DEADLOCK

MÔ HÌNH HÓA HỆ THỐNG

- Hệ thống gồm một tập các loại tài nguyên, kí hiệu R_1, R_2,..., R_m
  - Ví dụ: CPU cycles, memory space, I/O devices, .. .
- Mỗi loại tài nguyên *Ri* có một tập các thể hiện (instances) W_i
- Tiến trình sử dụng tài nguyên theo các bước:
  1. Yêu cầu (request) - phải chờ nếu không được đáp ứng ngay.
  2. Sử dụng (use)
  3. Giải phóng (release)
- Các tác vụ yêu cầu và hoàn trả được thực hiện bằng các lời gọi hệ thống.

ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN - RAG

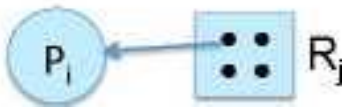
- Là đồ thị có hướng, với tập đỉnh V và tập cạnh E
- Tập đỉnh V gồm 2 loại:
  - Tập $P = \{P_1, P_2, \ldots, P_n\}$: tập các tiến trình trong hệ thống
  - Tập $R = \{R_1, R_2, \ldots, R_m\}$: tập các tài nguyên của hệ thống
- Tập cạnh cũng gồm 2 loại:
  - Cạnh yêu cầu (request edge): có hướng từ *Pi* đến *Rj*
  - Cạnh cấp phát (assignment edge): có hướng từ R_j đến P_i

KÝ HIỆU

- Tiến trình: Pi
- Loại tài nguyên (với 4 thể hiện):

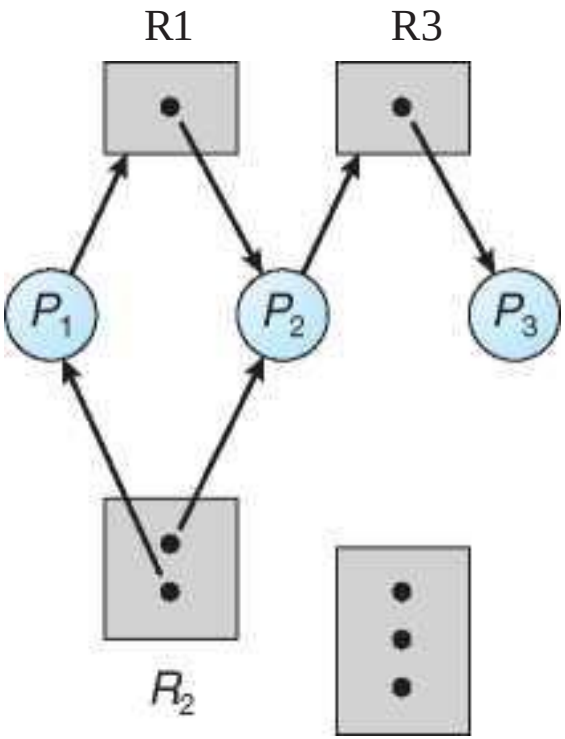
- *Pi* yêu cầu 1 thể hiện của *Rj*

- *Pi* đang giữ 1 thể hiện của *Rj*



ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN - VÍ DỤ 1

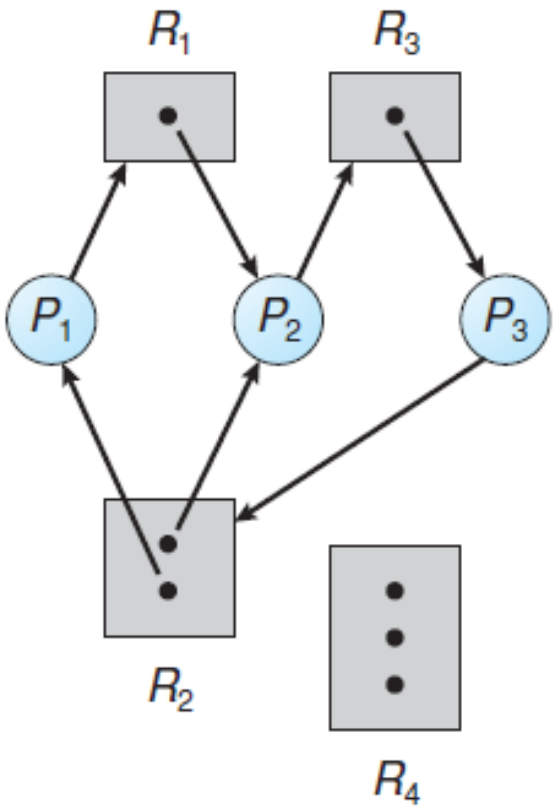
- Đồ thị cấp phát tài nguyên (không có chu trình và không deadlock):



- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$
- Thể hiện của các loại tài nguyên: $R_1 : 1; R_2 : 2; R_3 : 1; R_4 : 3.$
- Trạng thái của các tiến trình:
  - P_1: giữ $(R_2, 1)$; chờ $(R_1, 1)$
  - P_2: giữ $(R_1, 1)$, $(R_2, 1)$; chờ $(R_3, 1)$
  - P_3: giữ $(R_3, 1)$

ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN - VÍ DỤ 2

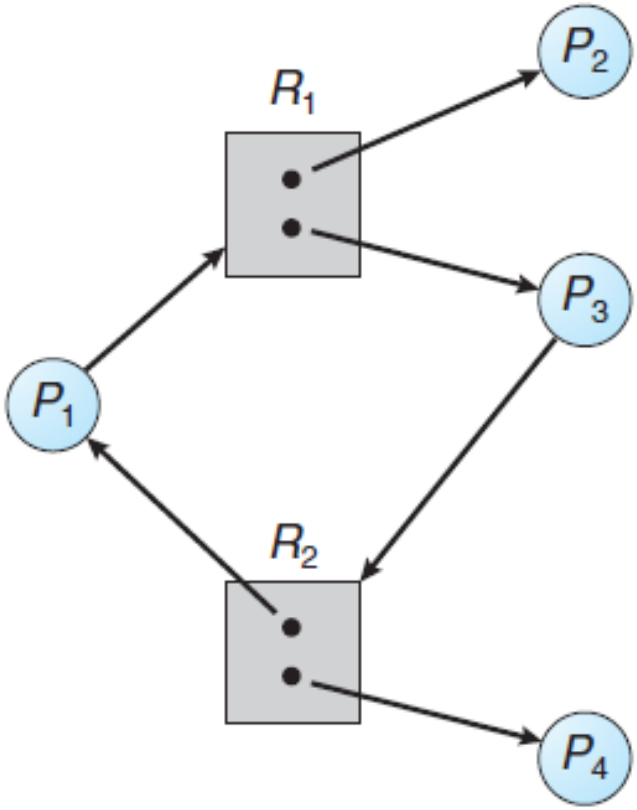
- Đồ thị cấp phát tài nguyên với chu trình và deadlock:



- Trạng thái của các tiến trình:
  - P_1: giữ $(R_2, 1)$; chờ $(R_1, 1)$
  - P_2: giữ $(R_1, 1)$, $(R_2, 1)$; chờ $(R_3, 1)$
  - P_3: giữ $(R_3, 1)$; chờ $(R_2, 1)$
- 2 chu trình nhỏ nhất (minimal cycles):
  $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
  $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$
- Deadlock: cả 3 tiến trình P_1, P_2, P_3

ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN - VÍ DỤ 3

- Đồ thị cấp phát tài nguyên có chu trình nhưng không deadlock:



- Chu trình:
  $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- Tại sao không deadlock?

ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN & DEADLOCK

- Nếu đồ thị không có chu trình: chắc chắn không có deadlock
- Nếu đồ thị có chu trình:
  - Nếu mỗi loại tài nguyên chỉ có một thể hiện: chắc chắn có deadlock.
  - Nếu mỗi loại tài nguyên có nhiều thể hiện: có thể có deadlock.

CÁC CÁCH TIẾP CẬN ĐỐI VỚI VẤN ĐỀ DEADLOCK

1. **Đề ra các giao thức để đảm bảo cho hệ thống không bao giờ rơi vào trạng thái deadlock.**

2. **Cho phép hệ thống rơi vào trạng thái deadlock, sau đó sử dụng các giải thuật để phát hiện deadlock và phục hồi.**

3. **Không quan tâm và không xử lý vấn đề deadlock trong hệ thống**

   - Khá nhiều hệ điều hành sử dụng phương pháp này.
   - Tuy nhiên, nếu deadlock không được phát hiện và xử lý sẽ dẫn đến việc giảm hiệu suất của hệ thống. Cuối cùng, hệ thống có thể ngưng hoạt động và phải khởi động lại.

NGĂN CHẶN VÀ TRÁNH DEADLOCK

- Có hai biện pháp để đảm bảo hệ thống không rơi vào trạng thái deadlock:
  1. **Ngăn chặn deadlock** (deadlock prevention): không cho phép (ít nhất) một trong bốn điều kiện cần cho deadlock xảy ra.
  2. **Tránh deadlock** (deadlock avoidance): các tiến trình cần cung cấp thông tin về tài nguyên nó cần để hệ thống cấp phát tài nguyên một cách thích hợp.

NGĂN CHẶN DEADLOCK - ĐIỀU KIỆN 1

Ngăn một trong bốn điều kiện cần của deadlock - thắt chặt cách thức yêu cầu tài nguyên của tiến trình.

1. **Ngăn Loại trừ hỗ tương** (mutual exclusion):
   - Tài nguyên có thể chia sẻ: không cần thiết.
   - Tài nguyên không thể chia sẻ: không thể thực hiện được do một số tài nguyên không thể được chia sẻ đồng thời cho nhiều tiến trình.

NGĂN CHẶN DEADLOCK - ĐIỀU KIỆN 2

2. **Ngăn Chờ và giữ** (wait & hold): phải đảm bảo khi một tiến trình yêu cầu một tài nguyên, nó không đang giữ một tài nguyên nào khác.

- **Cách 1**: mỗi t/trình yêu cầu toàn bộ tài nguyên cần thiết một lần.
- **Cách 2**: khi yêu cầu tài nguyên, tiến trình không đang giữ bất kỳ tài nguyên nào. Nếu đang giữ thì phải trả lại trước khi yêu cầu.
- **Ví dụ**: xét một tiến trình P copy dữ liệu từ băng từ sang đĩa từ, sau đó sắp xếp dữ liệu trên đĩa từ rồi in kết quả ra máy in.
  - Cách 1: *P* → {băng từ, đĩa từ, máy in} ngay khi t/trình bắt đau.
  - Cách 2: P → {băng từ, đĩa từ} → P; P → {đĩa từ, máy in}
- **Khuyết điểm**: hiệu suất sử dụng t/nguyên thấp + có thể bị đói t/nguyên

NGĂN CHẶN DEADLOCK - ĐIỀU KIÊN 3

3. **Ngăn Không trưng dụng** (no prermption): nếu một tiến trình A có đang giữ tài nguyên và yêu cầu thêm tài nguyên đang giữ bởi tiến trình khác thì:
   - **Cách 1**: hệ thống lấy lại mọi tài nguyên A đang giữ và A sẽ khởi động lại khi cả tài nguyên cũ và mới sẵn sàng cho nó.
   - **Cách 2**: hệ thống xem tài nguyên mà A yêu cầu:
     - Nếu tài nguyên đó đang được giữ bởi một tiến trình B cũng đang đợi thêm tài nguyên, hệ thống sẽ lấy lại tài nguyên của B và cấp phát cho *A*.
     - Nếu tài nguyên đó đang được giữ bởi một tiến trình không đang đợi thêm tài nguyên thì A phải đợi (tài nguyên của A đang giữ sẽ bị thu hồi nếu có tiến trình khác cần - trường hợp 1).

NGĂN CHẶN DEADLOCK - ĐIỀU KIỆN 4

4. Ngăn Chờ đợi vòng tròn (circular wait):

- Mỗi tài nguyên sẽ được gán một thứ tự toàn cục.
- Các tiến trình khi yêu cầu tài nguyên phải theo đúng thứ tự.
- Thông thường, một hàm one-to-one được định nghĩa để thực hiện gán thứ tự toàn cục: F : *R* → N; với *R* là tập các tài nguyên.
  - Một tiến trình đầu tiên có thể yêu cầu bất kỳ tài nguyên *Ri* nào.
  - Sau đó, một yêu cầu tài nguyên *Rj* chỉ được đáp ứng nếu F(*Rj*) > *F* (*Ri*), hoặc nó phải trả lại tài nguyên *Ri* trước khi xin tài nguyên *Rj*.

TRÁNH DEADLOCK

- Cách tiếp cận tránh deadlock cho phép sử dụng tài nguyên hiệu quả hơn ngăn chặn deadlock, bằng cách:
  - Yêu cầu mỗi tiến trình khai báo số lượng tài nguyên tối đa cần để thực hiện công việc.
  - Giải thuật tránh deadlock sẽ kiểm tra trạng thái cấp phát tài nguyên để đảm bảo hệ thống không rơi vào trạng thái deadlock.
  - Trạng thái cấp phát tài nguyên được định nghĩa bởi số lượng tài nguyên còn lại, số tài nguyên đã được cấp phát, và yêu cầu tối đa của các tiến trình.
- **Các giải thuật**: giải thuật Đồ thị cấp phát tài nguyên và giải thuật Banker.

TRẠNG THÁI AN TOÀN (SAFE STATE)

- Một hệ thống ở trạng thái an toàn nếu tồn tại một chuỗi an toàn.
- Một chuỗi tiến trình ($P_1, P_2, \ldots, P_n$) là một chuỗi an toàn nếu:
  - Với mọi P_i, $i = 1..n$, mọi yêu cầu về tài nguyên của P_i có thể được thỏa mãn bởi các tài nguyên đang sẵn sàng hoặc các tài nguyên đang bị giữ bởi Pj, với j < i ngăn chờ đợi vòng tròn).
- Một hệ thống ở trạng thái không an toàn nếu không tồn tại chuỗi an toàn.

VÍ DỤ VỀ CHUỖI AN TOÀN

- Cho một hệ thống có 12 băng từ và 3 tiến trình p_0, p_1, p_2.
- Trạng thái và nhu cầu sử dụng tài nguyên của 3 tiến trình tại thời điểm t_o được cho trong bảng sau:

	Maximum Needs	Current Needs
Po	10	5
P1	4	2
P2	9	2

- Chuỗi (P_1, P_0, p_2) là chuỗi an toàn → hệ thống an toàn.
- Giả sử tại thời điểm t1, p_2 yêu cầu và được cấp phát 1 băng từ → hệ thống còn 2 băng từ sẵn sàng và hệ thống trở nên không an toàn. (?)

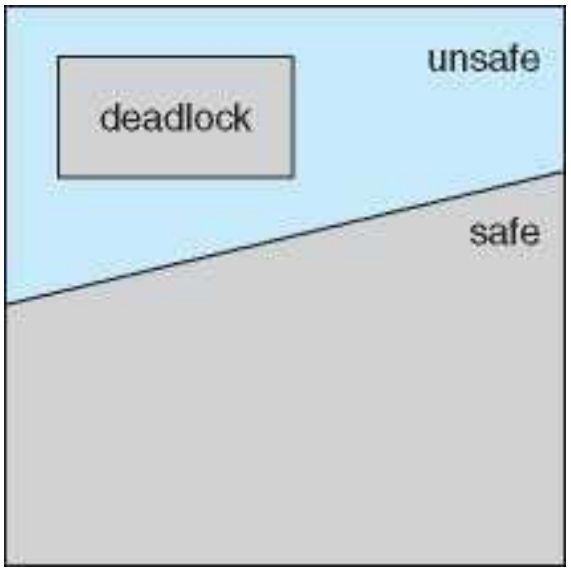
TRẠNG THÁI AN TOÀN VÀ DEADLOCK

- Nếu hệ thống đang trong trạng thái an toàn → không deadlock.
- Nếu hệ thống đang ở trạng thái không an toàn → có thể có deadlock.

- **Ý tưởng cho giải pháp tránh deadlock**:

đảm bảo hệ thống không rơi vào trạng thái không an toàn bằng cách:

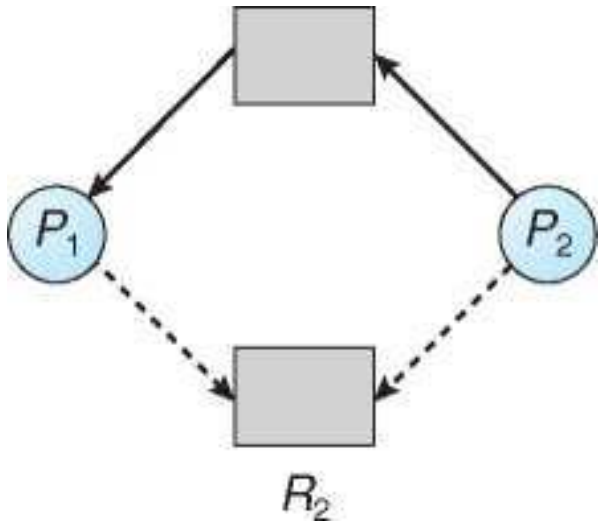
- **Khi một tiến trình yêu cầu một tài nguyên đang sẵn sàng, hệ thống sẽ kiểm tra nếu việc cấp phát không dẫn đến trạng thái không an toàn thì sẽ cấp phát ngay.**



NGĂN CHẶN VÀ TRÁNH DEADLOCK

GIẢI THUẬT ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN

- Được áp dụng cho hệ thống chỉ có 1 thể hiện cho mỗi loại tài nguyên.
- Bổ sung thêm 1 loại cạnh, "cạnh dự định" yêu cầu, *Pi* → *Rj*: *Pi* có thể yêu cầu *Rj*, được biểu diễn bằng đường ngắt khúc trên đồ thị.
- Việc yêu cầu tài nguyên phải được dự tính trước trong hệ thống → các cạnh dự định phải xuất hiện sẵn trong đồ thị.

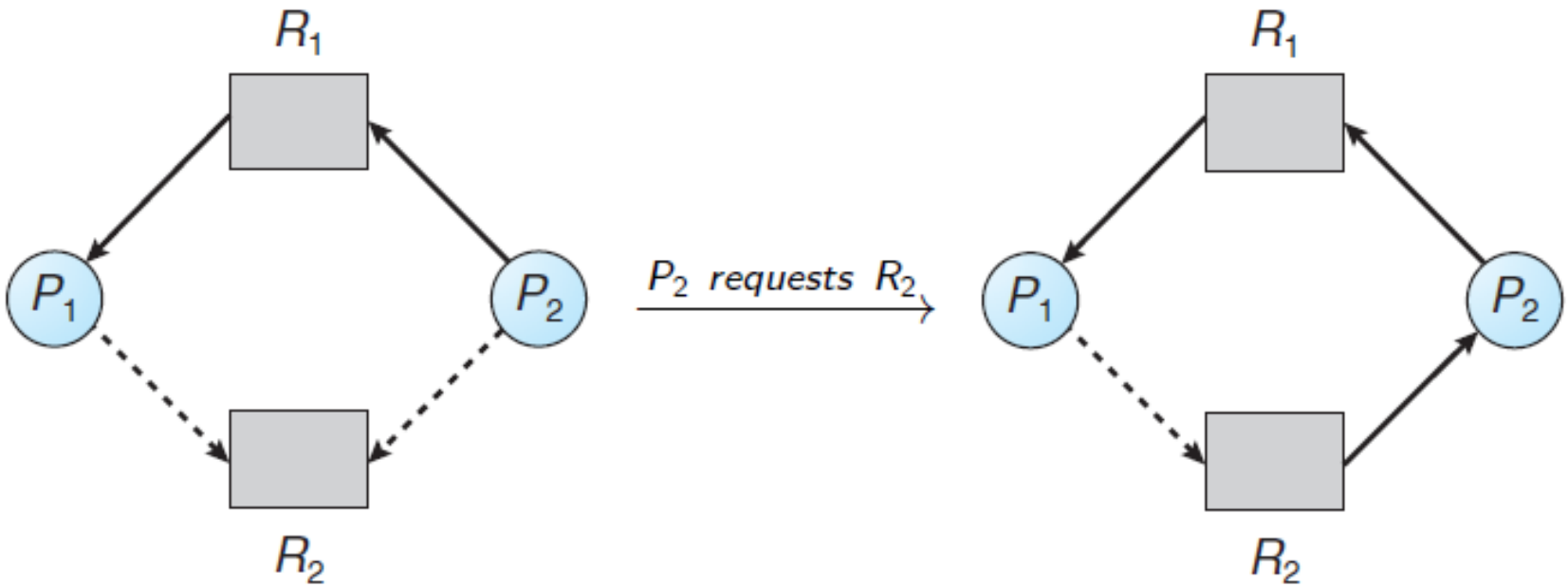


GIẢI THUẬT ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN

- Sự chuyển đổi của các cạnh trong đồ thị:
  - Khi tiến trình yêu cầu tài nguyên: cạnh dự định → cạnh yêu cầu.
  - Khi tài nguyên được cấp phát: cạnh yêu cầu → cạnh cấp phát .
  - Khi tài nguyên được giải phóng: cạnh cấp phát → cạnh dự định yêu cầu.
- **Nguyên tắc cấp phát tài nguyên**: một yêu cầu tài nguyên *Rj* của tiến trình P_i chỉ được cấp phát khi việc chuyển từ *Pi* → *Rj* (cạnh yêu cầu) sang *Rj* → *Pj* (cạnh cấp phát) không tạo thành chu trình trong đồ thị cấp phát tài nguyên.

GIẢI THUẬT ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN

- Ví dụ: Yêu cầu tài nguyên của P_2 đối với R_2 sẽ không được cấp phát – mặc dù R_2 đang sẵn dùng – vì nó có thể tạo ra chu trình (nếu sau đó P_1 lại yêu cầu R_2) $\Rightarrow$ có thể dẫn đến deadlock.



GIẢI THUẬT BANKER

- Áp dụng cho hệ thống có nhiều thể hiện cho mỗi loại tài nguyên.
- Dựa trên ý tưởng của các nghiệp vụ ngân hàng.
- Điều kiện:
  - **Mỗi tiến trình phải khai báo số lượng thể hiện tối đa của mỗi loại tài nguyên mà nó cần.**
  - **Khi tiến trình yêu cầu tài nguyên thì nó có thể phải đợi (ngay cả khi tài nguyên được yêu cầu đang sẵn sàng).**
  - **Khi một tiến trình đã có được đầy đủ tài nguyên thì phải hoàn trả không một khoảng thời gian hữu hạn.**

GIẢI THUẬT BANKER - CẤU TRÚC DỮ LIỆU

Cho *n* = số lượng tiến trình, *m* = số lượng các loại tài nguyên,

***P*: tập các tiến trình, *R*: tập các loại tài nguyên.**

- Avaiable: **vector có chiều dài m.**

  Available[j] = *k* → **có *k* thể hiện của** Rj **đang sẵn sàng.**

- Max: **ma trận n x m.**

  Max[i,j] = k → **tiến trình *Pi* yêu cầu tối đa** k **thể hiện của *Rj*.**

- Allocation: **ma trận n x m.**

  Allocation[i,j] = k → *Pi* **đã được cấp phát** k **thể hiện của *Rj*.**

- Need: **ma trận n x m.**

  Need[i,j] = k → ***Pi* có thể yêu cầu thêm** k **thể hiện của *Rj*.**

  (Need [i, j] = Max [i, j] — Allocation[i, j])

GIẢI THUẬT BANKER - TÌM CHUỖI AN TOÀN

1. Cho *Work* và *Finish* là hai vector độ dài là *m* và n. Khởi tạo:

   Work = Available

   Finish[i] = false, for i = 0..(n — 1)

2. Tìm *i* thỏa hai điều kiện:

   a. *Finish[i] = false*

   b. *Need[i] <= Work*

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

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

   Finish[i] = true

   Quay lại bước 2.

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

Cho hai vector *X*, *Y* có độ dài n:

$X < Y \Leftrightarrow X[i] <= Y[i]$, với mọi i = 1..n

GIẢI THUẬT BANKER - CẤP PHÁT TÀI NGUYÊN

- Cho $Request_i$ là vector yêu cầu của tiến trình P_i.
  $Request_i$ [j] = k & tiến trình P_i cần k thể hiện của R_j.

- Khi một tiến trình P_i yêu cầu tài nguyên, thực hiện các bước sau:

1. Nếu $Request_i$, <= $Need_i$, sang bước 2.
   Ngược lại, báo lỗi vì tiến trình yêu cầu quá định mức tối đa.
2. Nếu $Request_i$ <= *Available*, sang bước 3.
   Ngược lại, P_i phải chờ vì hiện không đủ tài nguyên để cấp phát.
3. *(next slide)*

GIẢI THUẬT BANKER - CẤP PHÁT TÀI NGUYÊN

3. Giả định cấp phát tài nguyên cho P_i bằng cách cập nhật trạng thái của hệ thống như sau:

$$Available = Available — Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i, = Need_i — Request_i,$$

Áp dụng giải thuật kiểm tra trạng thái an toàn cho trạng thái trên:

- Nếu hệ thống an toàn: cấp tài nguyên cho P_i.
- Ngược lại, P_i phải đợi và phục hồi lại trạng thái của hệ thống (i.e. nghịch đảo lại các cập nhật giả định bên trên).

GIẢI THUẬT BANKER - VÍ DỤ

- Cho một hệ thống có:
  - 5 tiến trình $P_0, \ldots, P_4$.
  - 3 loại tài nguyên: A (10 thể hiện), B (5 thể hiện), C (7 thể hiện).
- Trạng thái cấp phát tài nguyên tại thời điểm T_0:

	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2	7	4	3
P_1	2	0	0	3	2	2				1	2	2
P_2	3	0	2	9	0	2				6	0	0
P_3	2	1	1	2	2	2				0	1	1
P_4	0	0	2	4	3	3				4	3	1

GIẢI THUẬT BANKER - VÍ DỤ

- Dùng giải thuật Banker **để** k/tra tính an toàn của hệ thống tại T_0:

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

Banker

	Work		
i	A	B	C
	3	***3***	***2***
1	5	3	2
3	7	4	3
4	7	4	5
2	10	4	7
0	10	5	7

- Tồn tại chuỗi an toàn: (P_1, P_3, P_4, P_2, P_0) => hệ thống an toàn.

GIẢI THUẬT BANKER - VÍ DỤ

► G/sử tại thời điểm T_1, P_1 yêu cầu (1, 0, 2). Yêu cầu có được đáp ứng?

1. (*Request*$_1$ < *Available*) = *true* vì (1, 0, 2) < (3, 3, 2)
2. Trạng thái giả định tồn tại chuỗi an toàn (P_1, P_3, P_0, P_2, P_4) => hệ thống ở trạng thái an toàn => cấp phát ngay lập tức.

	Allocation			Need		
	A	B	C	A	B	C
P_0	0	1	0	7	4	3
P_1	**3**	**0**	**2**	**0**	**2**	**0**
P_2	3	0	2	6	0	0
P_3	2	1	1	0	1	1
P_4	0	0	2	4	3	1
Available = (2, 3, 0)						

Banker

Work			
i	A	B	C
	2	**3**	**0**
1	5	3	2
3	7	4	3
0	7	5	3
2	10	5	5
4	10	5	7

NGĂN CHẶN VÀ TRÁNH DEADLOCK

GIẢI THUẬT BANKER - BÀI TẬP

Tại T1

1. Yêu cầu (3, 3, 0) của P_4 có được đáp ứng không?

2. Yêu cầu (0, 2, 0) của Po có được đáp ứng không?

PHÁT HIỆN DEADLOCK

- Trong cách tiếp cận "phát hiện và phục hồi" đv vấn đề deadlock:
  - Chấp nhận cho deaclock xảy ra trong hệ thống.
  - Sử dụng các giải thuật để phát hiện deadlock.
  - Nếu có deadlock thì sẽ tiến hành phục hồi hệ thống, dùng 1 sơ đồ phục hồi thích hợp.
- Các giải thuật phát hiện deadlock thường sử dụng RAG.
- Có 2 loại giải thuật:
  - Cho trường hợp mỗi loại tài nguyên chỉ có 1 thể hiện.
  - Cho trường hợp mỗi loại tài nguyên có nhiều thể hiện.

PHÁT HIÊN VÀ PHỤC HỒI DEADLOCK

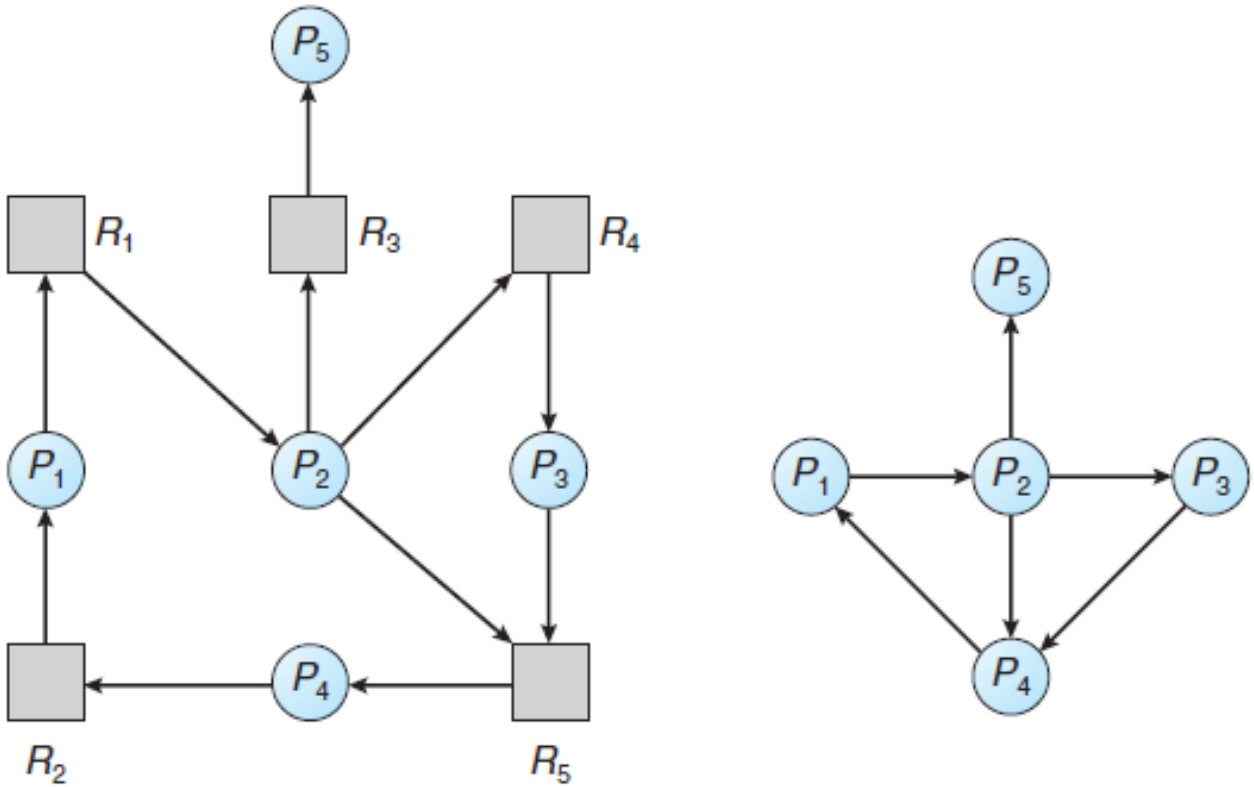
MỖI LOẠI TÀI NGUYÊN CHỈ CÓ 1 THỂ HIÊN

- Sử dụng 1 biến thể của RAG - đồ thị wait-for:
  - Các nút: là các tiến trình.
  - Các cạnh: $P_i \rightarrow Pj$ nếu P_i đang đợi P_j.
- Deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị wait-for tồn tại chu trình.
- Giải thuật xây dựng đồ thị wait-for và tìm kiếm chu trình sẽ được thực hiện định kỳ trong hệ thống.

PHÁT HIÊN VÀ PHỤC HỒI DEADLOCK

ĐỒ THỊ CẤP PHÁT TÀI NGUYÊN & WAIT-FOR

► **Một đồ thị Cấp phát tài nguyên và đồ thị Wait-for tương ứng:**



MỖI LOẠI TÀI NGUYÊN CÓ NHIỀU THỂ HIỆN

- Cấu trúc dữ liệu của giải thuật:
  - **Available**: vector có chiều dài m

    Số thể hiện còn sẵn dùng của mỗi loại tài nguyên.
  - **Allocation**: ma trận n x m

    Số thể hiện đã cấp phát cho mỗi tiến trình.
  - **Request**: ma trận n x m

    Yêu cầu tài nguyên hiện tại của các tiến trình.

Nếu *Request[i*, j] = k, P_i đang yêu cầu thêm *k* thể hiện của R_j.

PHÁT HIÊN VÀ PHỤC HỒI DEADLOCK

MỖI LOẠI TÀI NGUYÊN CÓ NHIỀU THỂ HIỆN

- Giải thuật phát hiện deadlock bao gồm 4 bước:
  1. Cho *Work* và *Finish* là 2 vector kích thước tương ứng là *m* và *n*. Khởi tạo:
     a. *Work = Available*
     b. for $i = 0, 1, \ldots, (n - 1)$

        if *Allocation[i]* <> 0 then

            Finish[i] = false

        else

            Finish[i] = true
  2. *(next slide)*

PHÁT HIỆN VÀ PHỤC HỒI DEADLOCK

MỖI LOẠI TÀI NGUYÊN CÓ NHIỀU THỂ HIÊN

2. Tìm i € [0, n — 1] thỏa mãn hai điều kiện sau:

   a. *Finish[i] = false*

   b. *Request[i] <= Work*

   Nếu không tồn tại, nhảy đến bước 4.

3. *Work = Work + Allocation[i]*
   Finish[i] = true
   Quay về bước 2.

4. Nếu tồn tại i với *Finish[i] = false*:

   - Hệ thống đang ở trạng thái deadlock.
   - Nếu *Finish[i] = false* ➔ P_i đang bị deadlock.

MỖI LOẠI TÀI NGUYÊN CÓ NHIỀU THỂ HIỆN

- Ví dụ 1: Cho hệ thống có
  - 5 tiến trình $P_0, \ldots, P_4$
  - 3 loại tài nguyên: A (7 thể hiện), B (2 thể hiện), C (6 thể hiện)
- Hiện trạng của hệ thống tại T_0:

	Allocation			Request		
	A	B	C	A	B	C
P_0	0	1	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
Available = (0, 0, 0)						

- Thực hiện giải thuật, ta tìm được chuỗi *(Po, P_2, P_1, P_3, P_4)* sẽ dẫn đến *Finish[i] = true* với mọi i = 0..4

=>hệ thống không deadlock tại T_0.

MỖI LOẠI TÀI NGUYÊN CÓ NHIỀU THỂ HIỆN

- Ví dụ 2 : Giả sử giữa thời điểm T_0 và T_1, P_2 yêu cầu thêm một thể hiện của tài nguyên loại C. Trạng thái của hệ thống tại T_1 là gì?

- Hiện trạng của hệ thống tại T_1.

	Allocation			Request		
	A	B	C	A	B	C
P_0	0	1	0	0	0	0
P_1	2	0	0	2	0	2
P_2	3	0	3	0	0	1
P_3	2	1	1	1	0	0
P_4	0	0	2	0	0	2
Available = (0, 0, 0)						

- Chạy giải thuật p/hiện deadlock:
- *Finish[0] = true*
- *Finish[l.4] = false*

=> Hệ thống deadlock (**P_1, P_2, P_3, P_4**)

- Nhận xét: tài nguyên thu hồi từ P_0 vẫn không đủ đáp ứng cho các t/trình khác.

SỬ DỤNG GIẢI THUẬT PHÁT HIỆN DEADLOCK

- Khi nào?/Tần suất thực hiện phát hiện deadlock phụ thuộc vào:
  1. Deadlock có thường xảy ra không?
  2. Có nhiều tiến trình tham gia vào deadlock hay không?
- Nếu kiểm tra deadlock thường xuyên trong hệ thống ít deadlock sẽ làm hao phí tài nguyên (CPU) của hệ thống.
- Tuy nhiên, nếu thực hiện phát hiện deadlock với tần suất thấp:
  - Nếu deadlock diễn ra thường xuyên: nhiều chu trình trong đồ thị → không biết chính xác tiến trình nào gây ra deadlock.
  - Nếu có nhiều tiến trình tham gia vào deadlock: hiệu suất sử dụng tài nguyên hệ thống giảm.

PHỤC HỒI DEADLOCK

- Các giải pháp để phục hồi deadlock:
  1. Bằng tay (manual): người sử dụng/điều hành sẽ giải quyết
  2. Phục hồi tự động bằng cách "phá" deadlock, có 2 giải pháp:
     - Ngưng tiến trình: ngưng (terminate) 1 hoặc một vài tiến trình tham gia vào deadlock.
     - Lấy lại tài nguyên: trưng dụng 1 hoặc một vài tài nguyên của các tiến trình bị deadlock.

PHÁT HIỆN VÀ PHỤC HỒI DEADLOCK

NGƯNG TIẾN TRÌNH

1. **Ngưng tất cả các tiến trình bị deadlock:**
   - Chi phí lớn, đặc biệt khi ngưng các tiến trình "già"
2. **Ngưng lần lượt từng tiến trình cho đến khi không còn deadlock:**
   - Kết hợp với giải thuật phát hiện deadlock → phải xét đến chi phí.
   - Các yếu tố chọn tiến trình cần ngưng:
     - Độ ưu tiên
     - Thời gian thực thi, thời gian còn lại
     - Loại tài nguyên tiến trình đang sử dụng
     - Tài nguyên cần thêm
     - Số lượng tiến trình sẽ bị ngưng
     - Loại tiến trình

PHÁT HIỆN VÀ PHỤC HỒI DEADLOCK

LẤY LẠI TÀI NGUYÊN

- Lần lượt lấy lại tài nguyên của các tiến trình, cấp phát cho tiến trình khác cho đến khi không còn deadlock.
- Các vấn đề khi thu hồi tài nguyên:
  - **Chọn "nạn nhân"**: chọn tài nguyên và tiến trình cần thu hồi (dựa trên số tài nguyên sở hữu, thời gian đã thực thi, . . .)
  - **Cuộn lại** (rollback):
    - cuộn lại đến một trạng thái an toàn, khởi động lại từ vị trí đó
    - đòi hỏi hệ thống phải lưu lại thông tin về trạng thái của các tiến trình đang thực thi.
  - **Đói tài nguyên**: tránh trình trạng một tiến trình luôn bị chọn làm nạn nhân.

TỔNG KẾT

- Dealock: là trạng thái các tiến trình chờ vòng tròn lẫn nhau, không thể tiến triển.
- Đồ thị cấp phát tài nguyên: mô hình trạng thái cấp phát, sử dụng tài nguyên của các tiến trình trong hệ thống.
- Có 2 cách tiếp cận đối với vấn đề deadlock:
  - Ngăn chặn và tránh deadlock: đảm bảo trạng thái deadlock không bao giờ xảy ra.
  - Phát hiện và khôi phục deadlock: cho phép deadlock xảy ra, dùng giải thuật kiem tra trạng thái deadlock theo định kỳ và thực hiện khôi phục deadlock nếu có.
- Các giải thuật ngăn chặn, tránh và phát hiện deadlock cơ bản dựa vào đồ thị cấp phát tài nguyên.