Chương 6. Deadlock

Mục tiêu: nắm được

1. Vấn đề deadlock trong hệ thống
2. Mô hình hệ thống
3. Điều kiện cần để xảy ra deadlock
4. Resource Allocation Graph (RAG)
5. Các phương pháp giải quyết deadlock
    1. Deadlock prevention
    2. Deadlock avoidance
    3. Deadlock detection
    4. Deadlock recovery

1. Vấn đề deadlock trong hệ thống

- *Tình huống:* một tập các process bị blocked, mỗi process giữ tài nguyên và đang chờ tài nguyên mà process khác trong tập đang giữ.

- Ví dụ 1
  - Giả sử hệ thống có 2 file trên đĩa.
  - P1 và P2 mỗi process đang mở một file và yêu cầu mở file kia.

- Ví dụ 2
  - Semaphore A và B, khởi tạo bằng 1

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

2. Mô hình hóa hệ thống

- Hệ thống gồm các loại *tài nguyên*, kí hiệu R_1, R_2,..., R_m , bao gồm:
  - CPU cycle, không gian bộ nhớ, thiết bị I/O, file, semaphore,...

  Mỗi loại tài nguyên R_i có W_i thực thể (instance).

- Process thường sử dụng tài nguyên theo thứ tự sau
  - *Yêu cầu* (request): process phải chờ nếu yêu cầu không được đáp ứng ngay
  - *Sử dụng* (use): process sử dụng tài nguyên
  - *Hoàn trả* (release): process hoàn trả tài nguyên

- Các tác vụ yêu cầu (request) và hoàn trả (release) đều là system call. Ví dụ
  - request/release device
  - open/close file
  - allocate/free memory
  - wait/signal

3. Điều kiện cần để xảy ra deadlock

Bốn điều kiện cần (necessary condition) để xảy ra deadlock

1. *Mutual exclusion:* ít nhất một tài nguyên được giữ theo nonsharable mode (ví dụ: printer; ví dụ sharable resource: read-only files).

2. *Hold and wait:* một process đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do tiến trình khác đang giữ.

3. Điều kiện cần để xảy ra deadlock (tt)

3. *No preemption:* (= no resource preemption) tài nguyên không thể bị lấy lại, mà chỉ có thể được trả lại từ process đang giữ tài nguyên đó khi nó muốn.

4. *Circular wait:* tồn tại một tập $\{P_0,...,P_n\}$ các tiến trình đang đợi sao cho

   P_0 đợi một tài nguyên mà P_1 đang giữ

   P_1 đợi một tài nguyên mà P_2 đang giữ

   ...

   P_n đợi một tài nguyên mà P_0 đang giữ

4. Resource Allocation Graph

- *Resource allocation graph* (RAG) là đồ thị có hướng, với tập đỉnh V và tập cạnh E

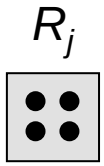
  - Tập đỉnh V gồm 2 loại:
    - $P = \{P_1, P_2, ..., P_n\}$ (Tất cả process trong hệ thống)
    - $R = \{R_1, R_2, ..., R_m\}$ (Tất cả các loại tài nguyên trong hệ thống)

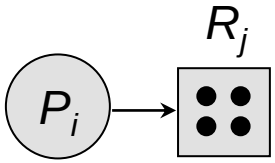
  - Tập cạnh E gồm 2 loại:
    - *Request edge:* cạnh có hướng từ P_i đến R_j
    - *Assignment edge:* cạnh có hướng từ R_j đến P_i

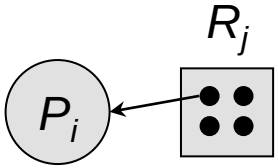
4. Resource Allocation Graph (tt)

Ký hiệu

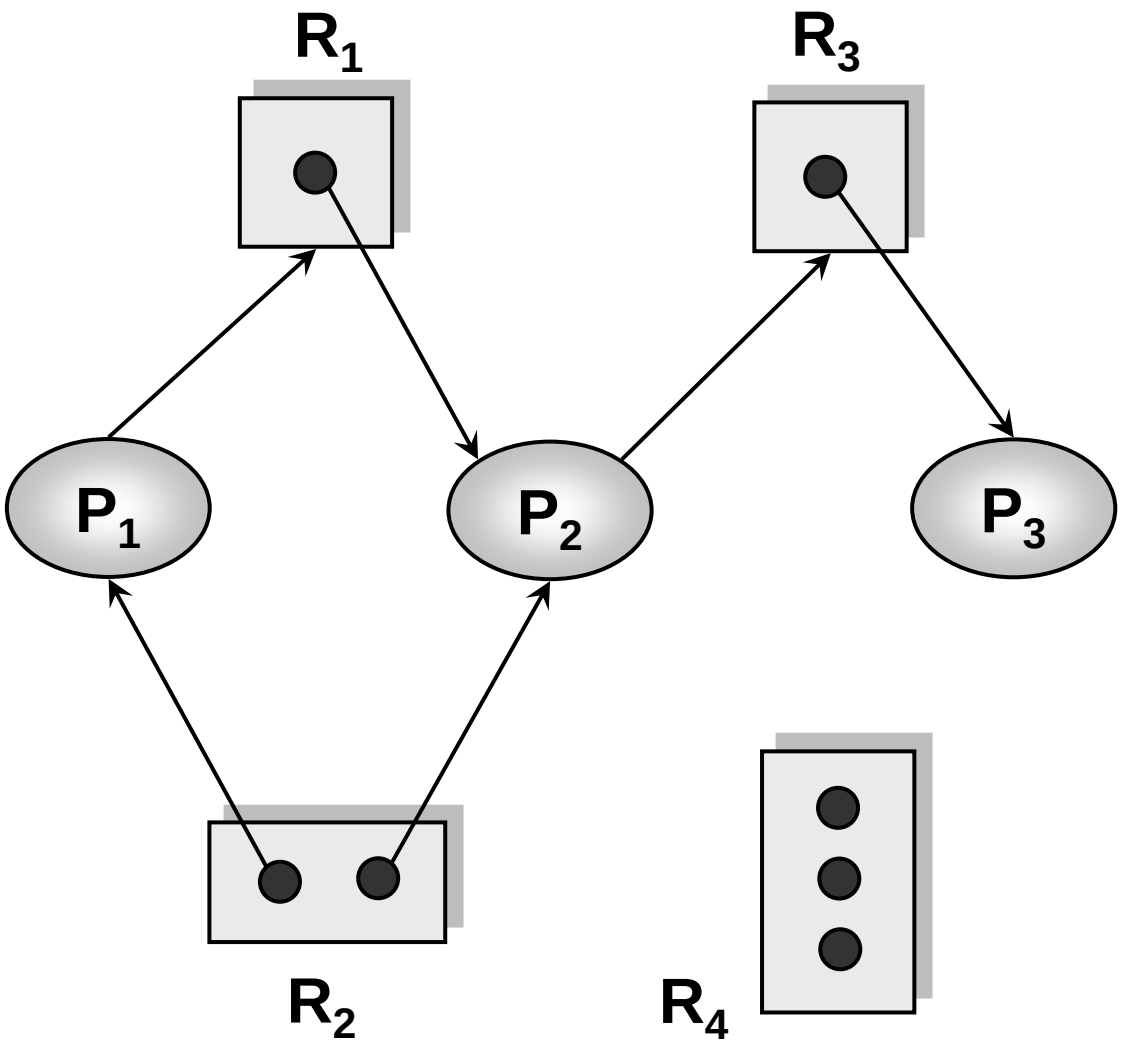
- Process: 

- Loại tài nguyên với 4 thực thể: 

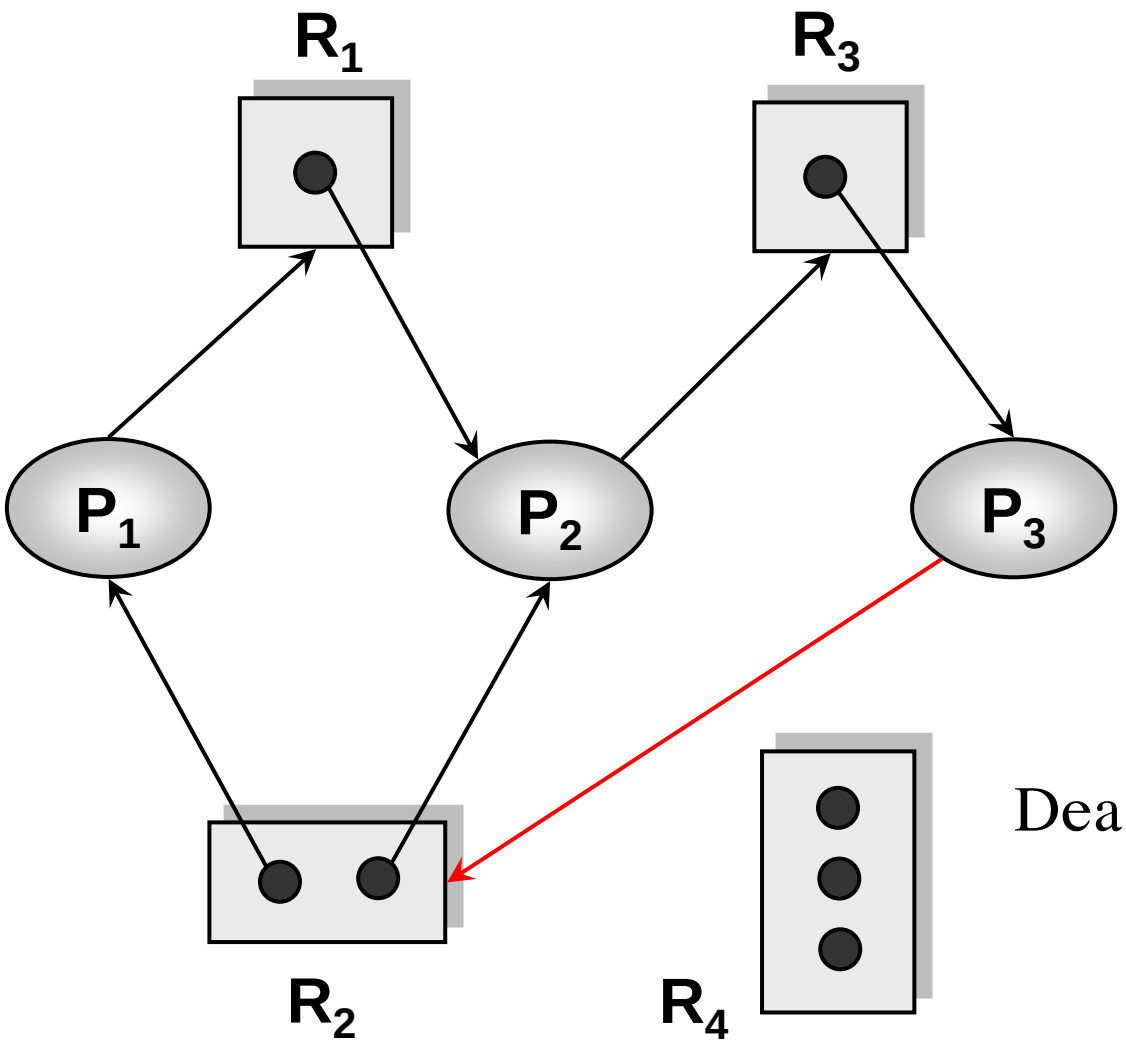
- P_i yêu cầu một thực thể của R_j : 

- P_i đang giữ một thực thể của R_j : 

Ví dụ về RAG



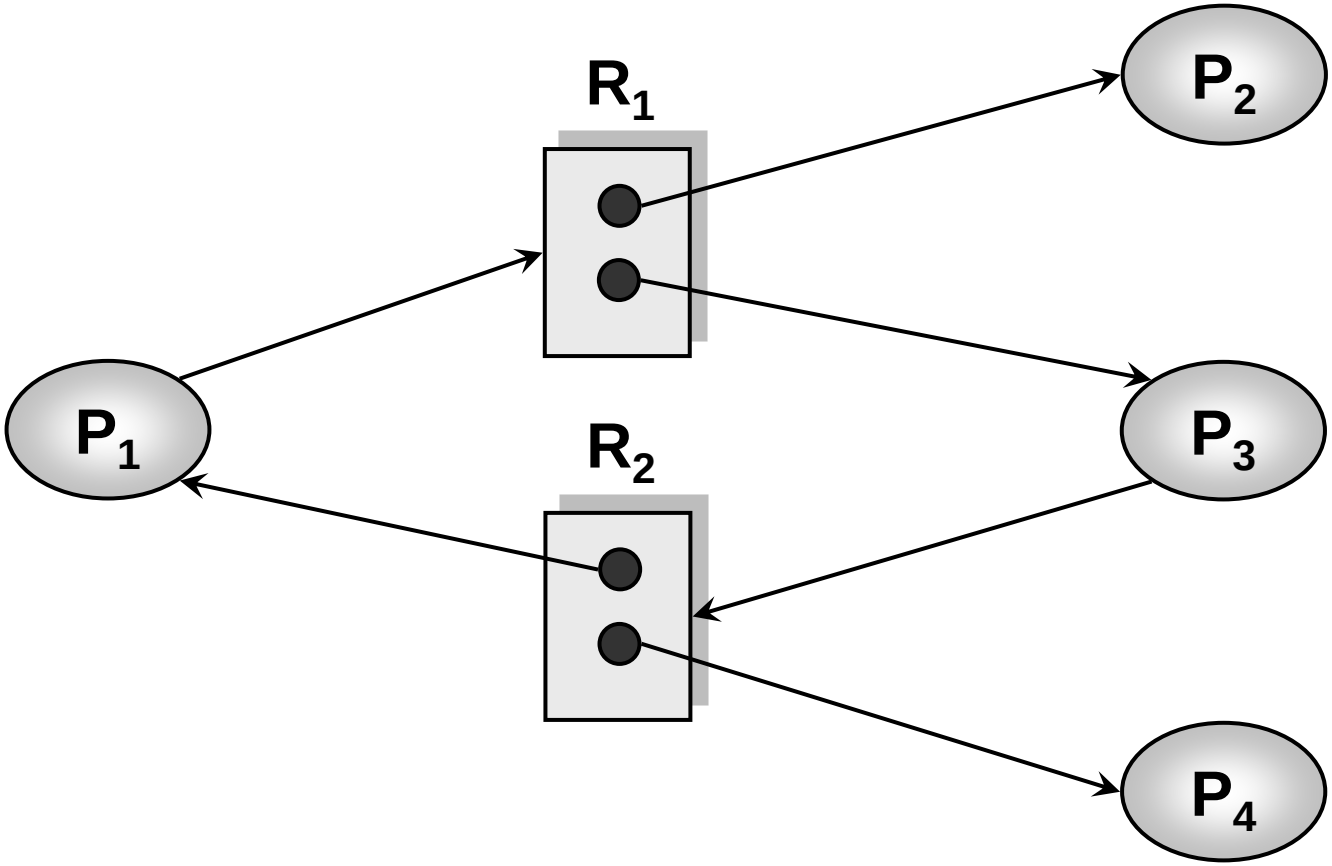
Ví dụ về RAG (tt)



Deadlock xảy ra!

RAG và deadlock

- Ví dụ một RAG chứa chu trình nhưng không xảy ra deadlock: P_4 có thể trả lại instance của R_2.



RAG và deadlock (tt)

- RAG không chứa chu trình (cycle) ⇒ không có deadlock
- RAG chứa một (hay nhiều) chu trình
  - Nếu mỗi loại tài nguyên chỉ có một thực thể ⇒ deadlock
  - Nếu mỗi loại tài nguyên có nhiều thực thể ⇒ có thể xảy ra deadlock

5. Các phương pháp giải quyết deadlock (1)

Ba phương pháp

1) Bảo đảm rằng hệ thống không rơi vào tình trạng deadlock bằng cách *ngăn* (preventing) hoặc *tránh* (avoiding) deadlock.

Khác biệt

- Ngăn deadlock: không cho phép (ít nhất) một trong 4 điều kiện cần cho deadlock
- Tránh deadlock: các tiến trình cần cung cấp thông tin về tài nguyên nó cần để hệ thống cấp phát tài nguyên một cách thích hợp

5. Các phương pháp giải quyết deadlock (2)

2) Cho phép hệ thống vào trạng thái deadlock, nhưng sau đó phát hiện deadlock và phục hồi hệ thống.

3) Bỏ qua mọi vấn đề, xem như deadlock không bao giờ xảy ra trong hệ thống.

☺Khá nhiều hệ điều hành sử dụng phương pháp này.

– Deadlock không được phát hiện, dẫn đến việc giảm hiệu suất của hệ thống. Cuối cùng, hệ thống có thể ngưng hoạt động và phải được khởi động lại.

5.1. Ngăn deadlock

Ngăn deadlock bằng cách ngăn một trong 4 điều kiện cần của deadlock

1. Ngăn mutual exclusion
   - đối với nonsharable resource (vd: printer): không làm được
   - đối với sharable resource (vd: read-only file): không cần thiết

5.1. Ngăn deadlock (tt)

2. Ngăn Hold and Wait

- Cách 1: mỗi process yêu cầu toàn bộ tài nguyên cần thiết một lần. Nếu có đủ tài nguyên thì hệ thống sẽ cấp phát, nếu không đủ tài nguyên thì process phải bị blocked.

- Cách 2: khi yêu cầu tài nguyên, process không được giữ bất kỳ tài nguyên nào. Nếu đang có thì phải trả lại trước khi yêu cầu.

- Ví dụ để so sánh hai cách trên: một tiến trình copy dữ liệu từ tape drive sang disk file, sắp xếp disk file, rồi in kết quả ra printer.

- Khuyết điểm của các cách trên:
  - Hiệu suất sử dụng tài nguyên (resource utilization) thấp
  - tiến trình có thể bị starvation

5.1. Ngăn deadlock (tt)

3. Ngăn No Preemption: nếu process A có giữ tài nguyên và đang yêu cầu tài nguyên khác nhưng tài nguyên này chưa cấp phát ngay được thì

   – Cách 1: Hệ thống lấy lại mọi tài nguyên mà A đang giữ
     - A chỉ bắt đầu lại được khi có được các tài nguyên đã bị lấy lại cùng với tài nguyên đang yêu cầu

   – Cách 2: Hệ thống sẽ xem tài nguyên mà A yêu cầu
     - Nếu tài nguyên được giữ bởi một process khác đang đợi thêm tài nguyên, tài nguyên này được hệ thống lấy lại và cấp phát cho A.
     - Nếu tài nguyên được giữ bởi process không đợi tài nguyên, A phải đợi và tài nguyên của A bị lấy lại. Tuy nhiên hệ thống chỉ lấy lại các tài nguyên mà process khác yêu cầu

5.1. Ngăn deadlock (tt)

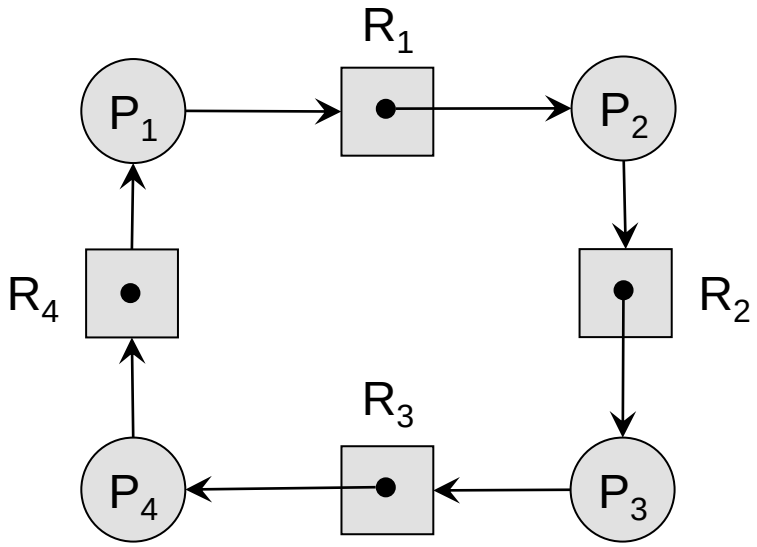
4. Ngăn Circular Wait: tập các loại tài nguyên trong hệ thống được gán một thứ tự hoàn toàn.

   – Ví dụ: F(tape drive) = 1, F(disk drive) = 5, F(printer) = 12
     - F là hàm định nghĩa thứ tự trên tập các loại tài nguyên.

5.1. Ngăn deadlock (tt)

4. Ngăn Circular Wait (tt)
   - Cách 1: mỗi process chỉ có thể yêu cầu thực thể của một loại tài nguyên theo thứ tự tăng dần (định nghĩa bởi hàm F) của loại tài nguyên. Ví dụ
     - Chuỗi yêu cầu thực thể hợp lệ: tape drive → disk drive → printer
     - Chuỗi yêu cầu thực thể không hợp lệ: disk drive → tape drive
   - Cách 2: Khi một process yêu cầu một thực thể của loại tài nguyên R_j thì nó phải trả lại các tài nguyên R_i với $F(R_i) > F(R_j)$.
   - "Chứng minh" cho cách 1: phản chứng
     - $F(R_4) < F(R_1)$
     - $F(R_1) < F(R_2)$
     - $F(R_2) < F(R_3)$
     - $F(R_3) < F(R_4)$

     Vậy $F(R_4) < F(R_4)$, mâu thuẩn!



5.1. Deadlock avoidance

- Deadlock prevention sử dụng tài nguyên không hiệu quả.
- Deadlock avoidance vẫn đảm bảo hiệu suất sử dụng tài nguyên tối đa đến mức có thể.
- Yêu cầu mỗi process khai báo số lượng tài nguyên tối đa cần để thực hiện công việc
- Giải thuật deadlock-avoidance sẽ kiểm tra *trạng thái cấp phát tài nguyên* (resource-allocation state) để bảo đảm hệ thống không rơi vào deadlock.

  Trạng thái cấp phát tài nguyên được định nghĩa dựa trên số tài nguyên còn lại, số tài nguyên đã được cấp phát và yêu cầu tối đa của các process.

Trạng thái safe và unsafe

- Một trạng thái của hệ thống được gọi là *an toàn* (safe) nếu tồn tại một *chuỗi an toàn* (safe sequence).

Chuỗi an toàn

- Một chuỗi tiến trình <P_1, P_2,..., P_n > là một *chuỗi an toàn* nếu
  - Với mọi i = 1,...,n, yêu cầu tối đa về tài nguyên của P_i có thể được thỏa bởi
    - tài nguyên mà hệ thống đang có sẵn sàng (available)
    - cùng với tài nguyên mà tất cả P_j , j < i, đang giữ.

- Một trạng thái của hệ thống được gọi là *không an toàn* (unsafe) nếu không tồn tại một chuỗi an toàn.

Chuỗi an toàn (tt)

Ví dụ: Hệ thống có 12 tape drives và 3 tiến trình P_0, P_1, P_2

- Tại thời điểm t_0

	cần tối đa	đang giữ
P_0	10	5
P_1	4	2
P_2	9	2

- Còn 3 tape drive sẵn sàng.
- Chuỗi <P_1, P_0, P_2> là chuỗi an toàn $\Rightarrow$ hệ thống là an toàn

Chuỗi an toàn (tt)

- Giả sử tại thời điểm t_1, P_2 yêu cầu và được cấp phát 1 tape drive
  - còn 2 tape drive sẵn sàng

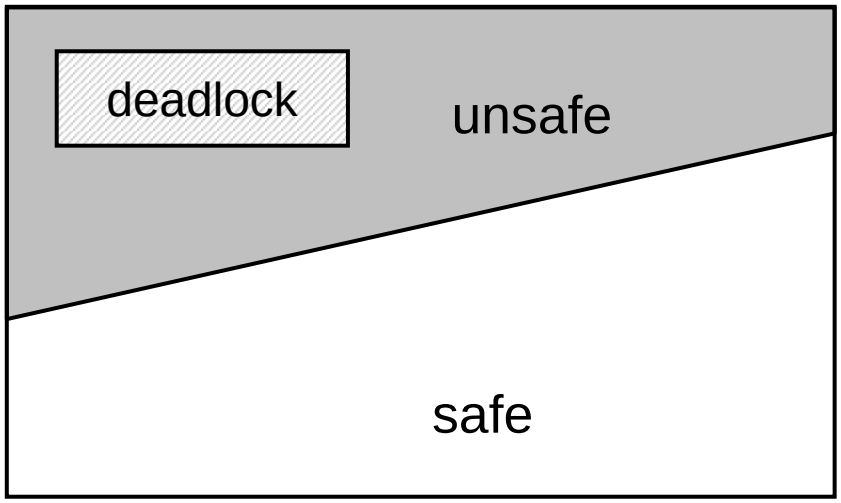
	cần tối đa	đang giữ
P_0	10	5
P_1	4	2
P_2	9	3

Hệ thống trở nên không an toàn.

- Khi một process yêu cầu một tài nguyên đang sẵn sàng, hệ thống sẽ kiểm tra: nếu việc cấp phát này không dẫn đến tình trạng unsafe thì sẽ cấp phát ngay.

Trạng thái safe/unsafe và deadlock

- Nếu hệ thống đang ở trạng thái safe ⇒ không deadlock.
- Nếu hệ thống đang ở trạng thái unsafe ⇒ có thể dẫn đến deadlock.
- Tránh deadlock bằng cách bảo đảm hệ thống không đi đến trạng thái unsafe.



Giải thuật banker

- Áp dụng cho hệ thống cấp phát tài nguyên trong đó mỗi loại tài nguyên có thể có nhiều instance.

- Bắt chước nghiệp vụ ngân hàng (banking)

- Điều kiện
  - Mỗi process phải khai báo số lượng thực thể (instance) tối đa của mỗi loại tài nguyên mà nó cần
  - Khi process yêu cầu tài nguyên thì có thể phải đợi mặc dù tài nguyên được yêu cầu đang có sẵn
  - Khi process đã có được đầy đủ tài nguyên thì phải hoàn trả trong một khoảng thời gian hữu hạn nào đó.

Giải thuật banker (tt)

n: số process, *m:* số loại tài nguyên

Các cấu trúc dữ liệu

Available: vector độ dài *m*

Available[j] = *k* ⇔ loại tài nguyên R_j có *k* instance sẵn sàng

Max: ma trận n × m

Max[i, j] = k ⇔ tiến trình P_i yêu cầu tối đa k instance của loại tài nguyên R_j

Allocation: ma trận n × m

Allocation[i, j] = k ⇔ Pi đã được cấp phát k instance của Rj

Need: ma trận n × m

Need[i, j] = k ⇔ Pi cần thêm k instance của Rj

Nhận xét: Need[i, j] = Max[i, j] – Allocation[i, j]

Ký hiệu Y ≤ X ⇔ Y[i] ≤ X[i], ví dụ (0, 3, 2, 1) ≤ (1, 7, 3, 2)

Giải thuật kiểm tra trạng thái an toàn

Tìm một chuỗi an toàn

1. Gọi Work và Finish là hai vector độ dài là m và n. Khởi tạo
   Work := Available
   Finish[i] := false, i = 1,..., n
2. Tìm i thỏa
   (a) Finish[i] = false
   (b) $Need_i$ ≤ Work (hàng thứ i của Need)
   Nếu không tồn tại i như vậy, đến bước 4.
3. Work := Work + $Allocation_i$
   Finish[i] := true
   quay về bước 2.
4. Nếu Finish[i] = true, i = 1,..., n, thì hệ thống đang ở trạng thái safe

Thời gian chạy của giải thuật là $O(mn^2)$

Giải thuật cấp phát tài nguyên

Gọi $Request_i$ là request vector của process P_i .

$Request_i [j] = k \Leftrightarrow P_i$ cần k instance của tài nguyên R_j .

1. Nếu $Request_i \leq Need_i$ thì đến bước 2. Nếu không, báo lỗi vì process đã vượt yêu cầu tối đa.

2. Nếu $Request_i \leq Available$ thì qua bước 3. Nếu không, P_i phải chờ vì tài nguyên không còn đủ để cấp phát.

Giải thuật cấp phát tài nguyên (tt)

3. Giả định cấp phát tài nguyên đáp ứng yêu cầu của P_i bằng cách cập nhật trạng thái 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 lên trạng thái trên

- Nếu trạng thái là safe thì tài nguyên được cấp thực sự cho P_i .
- Nếu trạng thái là unsafe thì P_i phải đợi, và phục hồi trạng thái:

$$Available := Available + Request_i$$
$$Allocation_i := Allocation_i - Request_i$$
$$Need_i := Need_i + Request_i$$

Giải thuật kiểm tra trạng thái an toàn – Ví dụ

- Có 5 process P_0 ,..., P_4
- Có 3 loại tài nguyên: A (có 10 instance), B (5 instance) và C (7 instance).
- Sơ đồ cấp phát trong hệ thống 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	❺
P_1	2	0	0	3	2	2				1	2	2	❶
P_2	3	0	2	9	0	2				6	0	0	❹
P_3	2	1	1	2	2	2				0	1	1	❷
P_4	0	0	2	4	3	3				4	3	1	❸

GT kiểm tra trạng thái an toàn – Vd (tt)

Chuỗi an toàn <P_1, P_3, P_4, P_2, P_0>

	Allocation	Need	Work
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	3 3 2
P_1	2 0 0	1 2 2	5 3 2
P_2	3 0 2	6 0 0	7 4 3
P_3	2 1 1	0 1 1	7 4 5
P_4	0 0 2	4 3 1	10 4 7 → 10 5 7

GT cấp phát tài nguyên – Ví dụ

- Yêu cầu (1, 0, 2) của P_1 có thỏa được không?
  - Kiểm tra điều kiện $Request_1 \leq Available$:
    - $(1, 0, 2) \leq (3, 3, 2)$ là đúng
  - Giả định thỏa yêu cầu, kiểm tra trạng thái mới có phải là safe hay không.

	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	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

  - Trạng thái mới là safe (chuỗi an toàn là $<P_1, P_3, P_4, P_0, P_2>$), vậy có thể cấp phát tài nguyên cho P_1.

GT cấp phát tài nguyên – Ví dụ (tt)

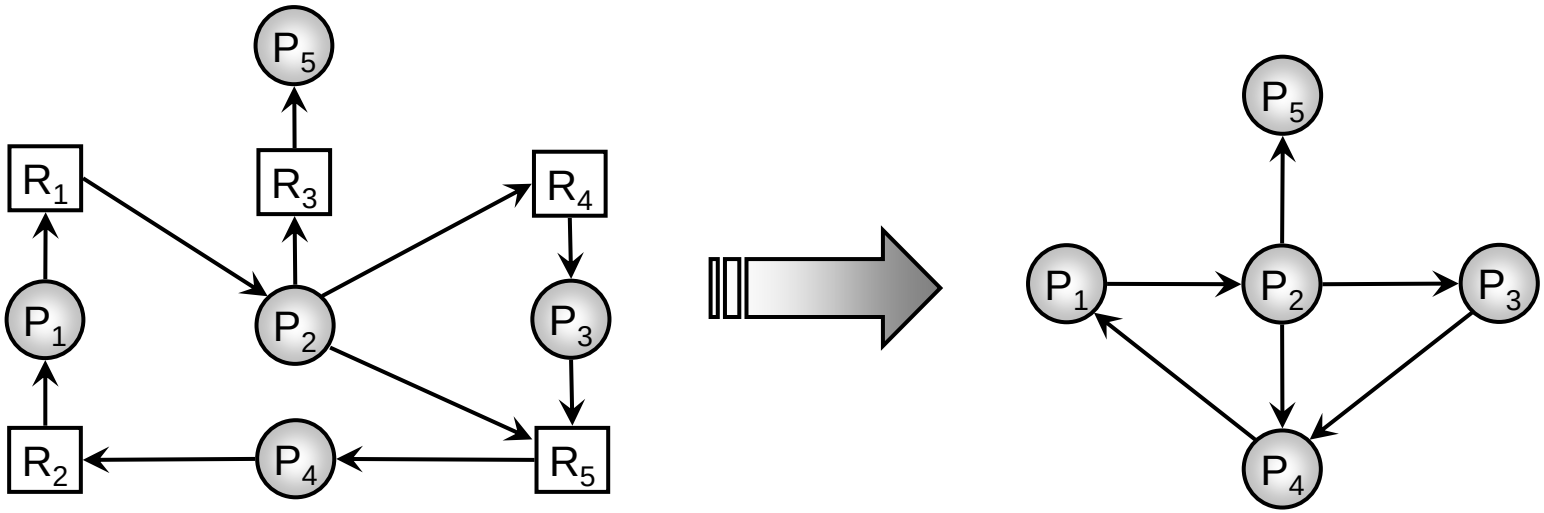
- P_4 yêu cầu (3, 3, 0) hoặc P_0 yêu cầu (0, 2, 0) thì có thỏa mãn được hay không?

5.2. Phát hiện deadlock

- Chấp nhận xảy ra deadlock trong hệ thống, kiểm tra trạng thái hệ thống bằng giải thuật phát hiện deadlock.
  - Nếu có deadlock thì tiến hành phục hồi hệ thống
- Các giải thuật phát hiện deadlock thường sử dụng mô hình RAG.
- Hệ thống cấp phát tài nguyên được khảo sát trong mỗi trường hợp sau
  1. Mỗi loại tài nguyên chỉ có một thực thể (instance)
  2. Mỗi loại tài nguyên có thể có nhiều thực thể

Mỗi loại tài nguyên chỉ có một thực thể

- Sử dụng *wait-for graph*
  - Wait-for graph được dẫn xuất từ RAG bằng cách bỏ các node biểu diễn tài nguyên và ghép các cạnh tương ứng.
    - Có cạnh từ P_i đến $P_j \Leftrightarrow P_i$ đang chờ tài nguyên từ P_j



- Một giải thuật kiểm tra có tồn tại chu trình trong wait-for graph hay không sẽ được gọi định kỳ. Giải thuật phát hiện chu trình có thời gian chạy là $O(n^2)$, với n là số đỉnh của graph.

Mỗi loại tài nguyên có nhiều thực thể

- Phương pháp dùng wait-for graph không áp dụng được cho trường hợp mỗi loại tài nguyên có nhiều instance.

- Các cấu trúc dữ liệu dùng trong giải thuật phát hiện deadlock

Available: vector độ dài m

số instance sẵn sàng của mỗi loại tài nguyên

Allocation: ma trận n × m

số instance của mỗi loại tài nguyên đã cấp phát cho mỗi process

Request: ma trận n × m

yêu cầu hiện tại của mỗi process.

Request [i, j] = k ⇔ Pi đang yêu cầu thêm k instance của Rj

Giải thuật phát hiện deadlock

1. Gọi *Work* và *Finish* là vector kích thước *m* và *n*. Khởi tạo:

   Work := Available

   i = 1, 2,..., n, nếu $Allocation_i \neq 0$ thì Finish[i] := false

   còn không thì Finish[i] := true

2. Tìm i thỏa mãn:

   Finish[i] := false và

   $Request_i \leq Work$

   Nếu không tồn tại i như thế, đến bước 4.

thời gian chạy của giải thuật

$O(m \cdot n^2)$

3. Work := Work + $Allocation_i$

   Finish[i] := true

   quay về bước 2.

4. Nếu Finish[i] = false, với một i = 1,..., n, thì hệ thống đang ở trạng thái deadlock. Hơn thế nữa, Finish[i] = false thì P_i deadlocked.

Giải thuật phát hiện deadlock – Ví dụ

- Hệ thống có 5 tiến trình P_0 ,..., P_4
  3 loại tài nguyên: *A* (7 instance), *B* (2 instance), *C* (6 instance).

	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			

Chạy giải thuật, tìm được chuỗi <P_0, P_2, P_3, P_1, P_4> với Finish[i] = true, i = 1,..., n, vậy hệ thống không deadlocked.

Giải thuật phát hiện deadlock – Ví dụ (tt)

- P_2 yêu cầu thêm một instance của C. Ma trận Request như sau:

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

  - Trạng thái của hệ thống là gì?
    - Vi phạm b*ước* 2 ($Request_i \leq Work$)
    - Có thể thu hồi tài nguyên đang sở hữu bởi process P_0 nhưng vẫn không đủ đáp ứng yêu cầu của các process khác.

      Vậy tồn tại deadlock, bao gồm các process P_1, P_2, P_3, và P_4.

5.3. Deadlock Recovery

- Khi deadlock xảy ra, để phục hồi
  - báo người vận hành (operator)

  hoặc
  - hệ thống tự động phục hồi bằng cách bẻ gãy chu trình deadlock:
    - chấm dứt một hay nhiều tiến trình
    - lấy lại tài nguyên từ một hay nhiều tiến trình

5.3. Deadlock Recovery: Chấm dứt tiến trình

- Phục hồi hệ thống bị deadlock bằng cách chấm dứt tiến trình
  - Chấm dứt tất cả process bị deadlocked, hoặc
  - Chấm dứt lần lượt từng process cho đến khi không còn deadlock
    - Sử dụng giải thuật phát hiện deadlock để xác định còn deadlock hay không

- Dựa trên yếu tố nào để chọn process cần được chấm dứt?
  - Độ ưu tiên của process
  - Thời gian đã thực thi của process và thời gian còn lại
  - Loại tài nguyên mà process đã sử dụng
  - Tài nguyên mà process cần thêm để hoàn tất công việc
  - Số lượng process cần được chấm dứt
  - Process là interactive process hay batch process

5.3. Deadlock Recovery: Lấy lại tài nguyên

- Lấy lại tài nguyên từ một process, cấp phát cho process khác cho đến khi không còn deadlock nữa.

- Các vấn đề trong chiến lược thu hồi tài nguyên:
  - Chọn "nạn nhân" để tối thiểu chi phí (có thể dựa trên số tài nguyên sở hữu, thời gian CPU đã tiêu tốn,...)
  - Rollback: rollback process bị lấy lại tài nguyên trở về trạng thái safe, tiếp tục process từ trạng thái đó. Hệ thống cần lưu giữ một số thông tin về trạng thái các process đang thực thi.
  - Starvation: để tránh starvation, phải bảo đảm không có process sẽ luôn luôn bị lấy lại tài nguyên mỗi khi deadlock xảy ra.

BÀI TẬP

1. Tìm trạng thái của hệ thống sau

Process	Max			Allocation			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	3	2	2	1	0	0	4	1	2
P2	6	1	3	2	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

Process	Need			Allocations			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	2	2	2	1	0	0	4	1	2
P2	4	0	2	2	1	1			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Process	Need			Allocations			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	2	2	2	1	0	0	6	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Process	Need			Allocations			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	0	0	0	0	0	7	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

Process	Need			Allocations			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	0	0	0	0	0	9	3	4
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	4	2	0	0	0	2			

Process	Need			Allocations			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	0	0	0			

- Chuỗi cấp phát an tồn: <P2,P1,P3,P4>

2. Tìm trạng thái của hệ thống sau. Nếu quá trình P2 thay bằng yêu cầu (0,4,2) thì hệ thống cấp phát tức thời cho nó không ?

Pr	Max			Allocation			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	0	1	0	0	1	1	5	2
P2	1	7	5	1	0	0			
P3	0	6	5	0	6	3			
P4	0	6	5	0	0	1			

Pr	Need			Allocation			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	0	0	0	0	1	1	5	2
P2	0	4	2	1	0	0			
P3	0	0	2	0	6	3			
P4	0	6	4	0	0	1			

Chuỗi cấp phát an tồn : <P1,P3,P2,P4>

QUÁ TRÌNH	Need			Allocations			Available		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	0	0	0	0	0	2	11	7
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	0	0	0			

3. Tìm trạng thái của hệ thống sau

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Tổng kết

- Những vấn đề cần nắm
- Bài tập:

  5.1 List three examples of deadlocks that are not related to a computer-system environment.

  5.2 Suppose that a system is in an unsafe state. Show that it is possible for the processes to complete their execution without entering a deadlock state.

  5.3 Prove that the safety algorithm presented in Section 7.5.3 requires an order of $m \times n^2$ operations.

5.4 Consider a computer system that runs 5,000 jobs per month with no deadlock-prevention or deadlock-avoidance scheme. Deadlocks occur about twice per month, and the operator must terminate and rerun about 10 jobs per deadlock. Each job is worth about $2 (in CPU time), and the jobs terminated tend to be about half-done when they are aborted.

A systems programmer has estimated that a deadlock-avoidance algorithm (like the banker's algorithm) could be installed in the system with an increase in the average execution time per job of about 10 percent. Since the machine currently has 30-percent idle time, all 5,000 jobs per month could still be run, although turnaround time would increase by about 20 percent on average.

a. What are the arguments for installing the deadlock-avoidance algorithm?

b. What are the arguments against installing the deadlock-avoidance algorithm?

5.5 Can a system detect that some of its processes are starving? If you answer “yes,” explain how it can. If you answer “no,” explain how the system can deal with the starvation problem.

5.6 Consider the following resource-allocation policy. Requests and releases for resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then we check any processes that are blocked,waiting for resources. If they have the desired resources, then these resources are taken away from them and are given to the requesting process. The vector of resources for which the process is waiting is increased to include the resources that were taken away.

- For example, consider a system with three resource types and the vector Available initialized to (4,2,2). If process P0 asks for (2,2,1), it gets them. If P1 asks for (1,0,1), it gets them. Then, if P0 asks for (0,0,1), it is blocked (resource not available). If P2 now asks for (2,0,0), it gets the available one (1,0,0) and one thatwas allocated to P0 (since P0 is blocked). P0's Allocation vector goes down to (1,2,1) and its Need vector goes up to (1,0,1).

  a. Can deadlock occur? If you answer "yes", give an example. If you answer "no," specify which necessary condition cannot occur.

  b. Can indefinite blocking occur? Explain your answer.

5.7 Suppose that you have coded the deadlock-avoidance safety algorithm and now have been asked to implement the deadlock-detection algo- rithm. Can you do so by simply using the safety algorithm code and redefining Ma xi = Waitingi + Al l o c a t i o ni ,where Waitingi is a vector specifying the resources process i is waiting for, and Al l o c a t i o ni is as defined in Section 7.5? Explain your answer.

5.8 Is it possible to have a deadlock involving only one single process? Explain your answer.