HỆ ĐIỀU HÀNH

1

NỘI DUNG

- Chương 1: Tổng quan
- Chương 2: Quản lý tiến trình
- Chương 3: Deadlock
- Chương 4: Quản lý bộ nhớ
- Chương 5: Hệ thống file
- Chương 6: Quản lý nhập xuất

2

Chương 3

Deadlock

Nội dung

1. Đặc điểm sử dụng tài nguyên của các tiến trình
2. Tình trạng deadlock
3. Giải pháp xử lý

Tài liệu tham khảo

- Andrew S. Tanenbaum, Modern Operating Systems, Pearson, 2015
  - Chương 6
- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley, 2013
  - Chương 7
- William Stallings,Operating Systems: Internals and Design Principles, Pearson, 2015
  - Chương 6

5

5

3.1. Đặc điểm sử dụng tài nguyên của các tiến trình

- Trong môi trường multiprogramming nhiều tiến trình có thể yêu cầu cùng một tài nguyên để có thể thực thi.
- Có hai loại tài nguyên:
  - Preemptable: tài nguyên có thể chia sẻ giữa các tiến trình (vd: bộ nhớ)
  - Nonpreemptable: tài nguyên không thể chia sẻ giữa các tiến trình (vd: đầu ghi đĩa)
- Các bước tiến trình sử dụng tài nguyên :
  - Yêu cầu tài nguyên (request)
  - Sử dụng tài nguyên (use)
  - Giải phóng (trả) tài nguyên (release)

6

6

Đặc điểm sử dụng tài nguyên của các tiến trình (tt)

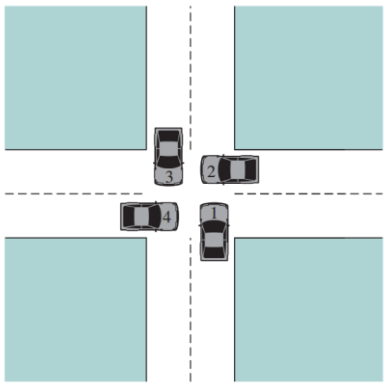
- Khi tiến trình yêu cầu tài nguyên:
  - Nếu tài nguyên có sẵn: sử dụng
  - Nếu tài nguyên không có sẵn: chuyển sang trạng thái chờ.
- Tài nguyên mà các tiến trình yêu cầu có thể là một tài nguyên không thể chia sẻ.
- Các tiến trình chờ tài nguyên có thể sẽ ko bao giờ thay đổi lại trạng thái được vì các tài nguyên mà nó yêu cầu đang bị giữ bởi các tiến trình khác → tắc nghẽn (deadlock)
- *Deadlock là tình trạng khi hệ thống tồn tại một tập hợp các tiến trình bị khóa, trong đó mỗi tiến trình đều đang chờ một sự kiện mà chỉ một tiến trình khác có thể tạo ra, và kết quả là không một tiến trình nào có thể hoàn thành*.

7

7

Đặc điểm sử dụng tài nguyên của các tiến trình (tt)

- Ví dụ: tình trạng kẹt xe



8

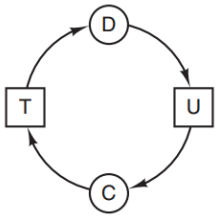
8

3.2. Tình trạng deadlock

- Điều kiện xảy ra deadlock
- Đồ thị cấp phát tài nguyên

Điều kiện xảy ra deadlock

- **Ngăn chặn lẫn nhau (Mutual exclusion)**: Một tài nguyên bị chiếm bởi một tiến trình, và không tiến trình nào khác có thể sử dụng tài nguyên này
- **Giữ và đợi (Hold-and-wait)**: Một tiến trình đang giữ ít nhất một tài nguyên và chờ một số tài nguyên khác đang bị một tiến trình khác chiếm giữ.
- **Không có đặc quyền (No-preemption)**: không thể thu hồi tài nguyên đã cấp cho tiến trình mà phải chờ tiến trình trả lại tài nguyên sau khi đã sử dụng xong
- **Chờ đợi vòng tròn (circular wait)**: Một tập tiến trình {P0, P1, ..., Pn}
  - P0 chờ một tài nguyên do P1 chiếm giữ
  - P1 chờ một tài nguyên khác do P2 chiếm giữ, ...,
  - Pn-1 chờ tài nguyên do Pn chiếm giữ
  - Pn chờ tài nguyên do P0 chiếm giữ



Đồ thị cấp phát tài nguyên (Resource allocation graph)

- Dùng để mô tả một cách chính xác tình trạng bế tắc
- Là một đồ thị có hướng G=(V, E) với V là tập đỉnh, E là tập các cạnh
- V được chia thành hai tập con
  - P= {P0, P1, ...,Pn} là tập các tiến trình trong hệ thống
  - R= {R0, R1, ..., Rm} là tập các loại tài nguyên trong hệ thống thỏa mãn **P∪R = E** và **P∩R= ∅**

11

11

Đồ thị cấp phát tài nguyên (tt)

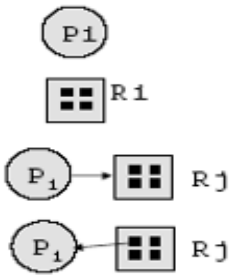
- Một cạnh có hướng từ Pi tới Rj (ký hiệu Pi →Rj): tiến trình Pi chờ tài nguyên Rj - gọi là **cạnh yêu cầu**
- Một cạnh có hướng từ Rj tới Pi (ký hiệu Rj → Pi): tài nguyên Rj đã được cấp phát cho tiến trình Pi - gọi là **cạnh gán**
- **Biểu diễn:**
  - Pi: hình tròn
  - Rj: hình chữ nhật. Rj có thể có nhiều thể hiện→biểu diễn mỗi thể hiện là một chấm nằm trong hình vuông.
  - Một cạnh yêu cầu trỏ tới chỉ một hình vuông Rj
  - Một cạnh gán gán tới một trong các dấu chấm trong hình vuông

12

12

Đồ thị cấp phát tài nguyên (tt)

- Khi Pi yêu cầu một thể hiện Rj, một cạnh yêu cầu được chèn vào đồ thị cấp phát tài nguyên.
- Khi yêu cầu được đáp ứng, cạnh yêu cầu được chuyển thành cạnh gán.

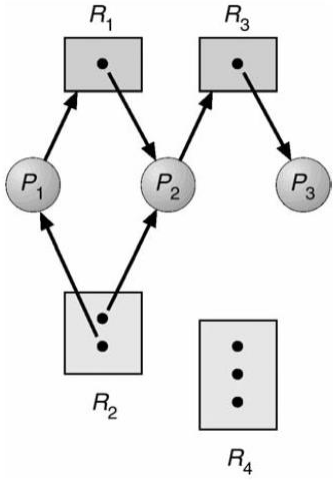


- Khi tiến trình không còn cần truy xuất tới tài nguyên, nó giải phóng tài nguyên → cạnh gán bị xoá.

13

Đồ thị cấp phát tài nguyên (tt)

- **Ví dụ**: các tập P, R, và E:
  - P = {P1, P2, P3}
  - R = {R1, R2, R3, R4}
  - E = {P1→R1, P2 →R3, R1 →P2, R2→P2, R3→P3}
- Trạng thái tiến trình:
  - P1 đang giữ một thể hiện của R2 và đang chờ một thể hiện của R1
  - P2 đang giữ một thể hiện của R1 và R2 và đang chờ một thể hiện của R3
  - P3 đang giữ một thể hiện của R3



14

Đồ thị cấp phát tài nguyên (tt)

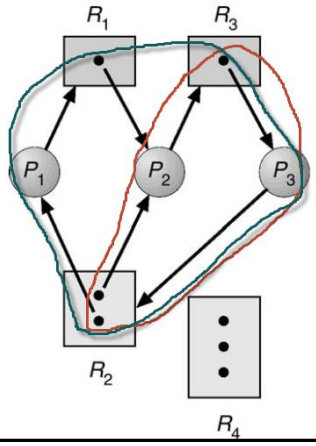
- Nếu đồ thị không chứa chu trình: **không** có deadlock.
- Nếu đồ thị có chứa chu trình: **có thể** có deadlock.
- Nếu một chu trình bao gồm chỉ một tập hợp các loại tài nguyên, mỗi loại tài nguyên chỉ có một thể hiện:
  - Có deadlock
  - Mỗi tiến trình chứa trong chu trình bị deadlock.
  - Một chu trình trong đồ thị là điều kiện cần và đủ để tồn tại deadlock.
- Nếu mỗi loại tài nguyên có nhiều thể hiện:
  - Chu trình trong đồ thị chưa chắc có deadlock.
  - Một chu trình trong đồ thị là điều kiện **cần nhưng chưa đủ** để tồn tại deadlock.

15

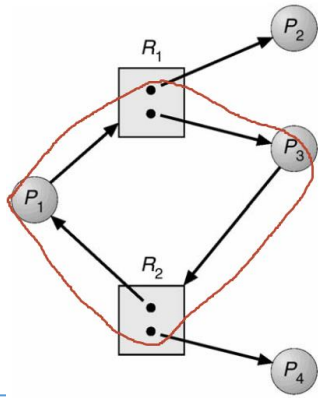
15

Đồ thị cấp phát tài nguyên (tt)

P3 yêu cầu một thể hiện của R2 → có deadlock
P1 → R1 → P2 → R3 → P3 → R2 → P1
P2 → R3 → P3 → R2 → P2



Có chu trình nhưng không có deadlock
P1 → R1 → P3 → R2 → P1

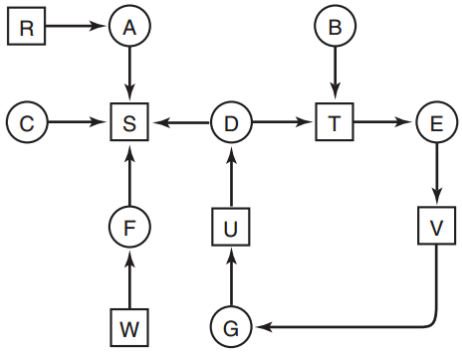


16

16

Đồ thị cấp phát tài nguyên (tt)

- Xét đồ thị cấp phát tài nguyên sau, cho biết hệ thống có xảy ra tình trạng deadlock hay không?

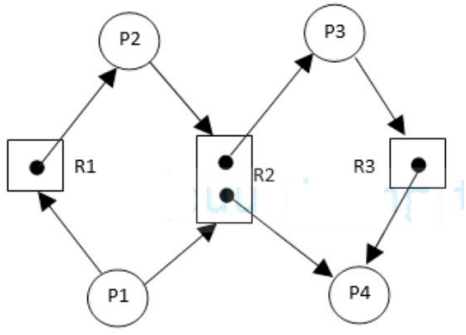


17

17

Đồ thị cấp phát tài nguyên (tt)

- Xét đồ thị cấp phát tài nguyên sau, cho biết hệ thống có xảy ra tình trạng deadlock hay không?



18

18

3.3. Giải pháp xử lý deadlock

1. Không quan tâm: có thể bỏ qua, xem như deadlock không bao giờ xảy ra trong hệ thống.
2. Ngăn chặn deadlock
3. Tránh xảy ra deadlocks
4. Phát hiện tình trạng deadlock và khôi phục.

19

19

3.3.1 Ngăn chặn deadlock

- Ngăn chặn một trong bốn điều kiện làm xảy ra deadlock
  - Loại trừ lẫn nhau (mutual exclusion)
  - Giữ và chờ cấp thêm tài nguyên (Hold and Wait)
  - Không đòi lại tài nguyên từ quá trình đang giữ chúng (No Preemption)
  - Tồn tại chu trình trong đồ thị cấp phát tài nguyên (Circular Wait)

20

20

Ngăn chặn mutual exclusion

- Đảm bảo hệ thống không có các tài nguyên không thể chia sẻ: gần như không tránh được

21

21

Ngăn chặn Giữ và chờ …

- Đảm bảo khi một tiến trình yêu cầu tài nguyên, nó không giữ bất cứ tài nguyên nào khác
  - Bắt buộc mỗi tiến trình phải yêu cầu toàn bộ tài nguyên cần thiết một lần, nếu có đủ tài nguyên hệ thống sẽ cấp phát, nếu không đủ tài nguyên, tiến trình sẽ bị block.
  - Khi yêu cầu tài nguyên, tiến trình không được giữ một tài nguyên khác, nếu có thì phải trả tài nguyên đang giữ trước khi yêu cầu.
- Hạn chế :
  - Hiệu quả sử dụng tài nguyên rất thấp
  - Có khả năng starvation

22

22

Ngăn chặn không có đặc quyền

- **Cách 1**: Nếu một tiến trình đang giữ một số tài nguyên và yêu cầu tài nguyên không có sẵn thì tiến trình phải trả tất cả tài nguyên hiện đang giữ
- **Cách 2**: nếu tài nguyên mà một tiến trình yêu cầu không có sẵn:
  - Nếu các tài nguyên đó đang được cấp phát tới các tiến trình khác đang chờ tài nguyên bổ sung → thu hồi lại →cấp cho tiến trình đang yêu cầu.
  - Nếu tài nguyên đó đang được cấp phát tới một tiến trình không đợi tài nguyên, tiến trình đang yêu cầu phải chờ và một số tài nguyên nó đang giữ có thể được đòi lại nếu có tiến trình khác yêu cầu.
- Được áp dụng cho tài nguyên mà trạng thái của nó có thể được lưu và cập nhật dễ dàng như các thanh ghi CPU, không gian bộ nhớ...

23

23

Ngăn chặn tạo chu trình trong đồ thị cấp phát tài nguyên

- Cấp phát tài nguyên theo một thứ tự phân cấp như sau:
  - Gọi R = {R1, R2, …, Rm} là tập hợp loại tài nguyên.
  - Đánh số thứ tự cho mỗi loại tài nguyên bằng cách định nghĩa hàm ánh xạ một-một F: R → N (N là tập hợp các số tự nhiên)
    - Ví dụ: F(ổ băng từ) = 1, F(đĩa từ) = 5, F(máy in) = 12
  - Một tiến trình đang giữ tài nguyên Ri thì chỉ được yêu cầu tài nguyên Rj nếu F(Rj) > F(Ri)

24

24

3.3.2 Tránh deadlock

- Sử dụng thuật toán cấp phát tài nguyên với đầu vào là các thông tin yêu cầu cấp phát tài nguyên của tiến trình để quyết định cấp phát tài nguyên sao cho deadlock không xảy ra.
- Có nhiều thuật toán tránh deadlock
- Thuật toán đơn giản:
  - Biết trước số thể hiện của mỗi loại tài nguyên mà mỗi tiến trình sẽ sử dụng.
  - → Hệ thống sẽ có đủ thông tin để xây dựng thuật toán cấp phát không gây ra bế tắc

25

25

Tránh deadlock (tt)

- Mục tiêu các thuật toán là kiểm tra trạng thái cấp phát tài nguyên "động" để đảm bảo điều kiện tạo chu trình chờ không xảy ra
- Trạng thái cấp phát tài nguyên được xác định bởi:
  - số lượng tài nguyên rỗi
  - số lượng tài nguyên đã cấp phát cho các tiến trình
  - số lượng tối đa tài nguyên mà mỗi tiến trình yêu cầu
- Hai thuật toán:
  - Thuật toán đồ thị cấp phát tài nguyên
  - Thuật toán banker

26

26

Khái niệm Trạng thái an toàn (safe-state)

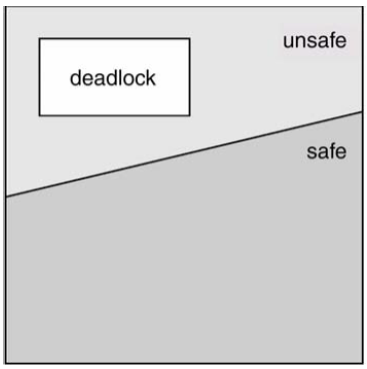
- Một trạng thái cấp phát tài nguyên được gọi là an toàn nếu hệ thống có thể cấp phát tài nguyên cho các tiến trình **theo một thứ tự nào đó** mà vẫn tránh được deadlock.
- Nói cách khác, một hệ thống ở trong trạng thái an toàn nếu và chỉ nếu ở đó tồn tại một thứ tự an toàn
- Thứ tự của các tiến trình <P1, P2, …, Pn> là một thứ tự an toàn khi: với mỗi thứ tự Pi, các tài nguyên mà Pi yêu cầu vẫn có thể được thoả mãn bởi:
  - tài nguyên hiện có + các tài nguyên được giữ bởi tất cả Pj, với j<i.

27

27

Trạng thái an toàn (tt)

- Trạng thái an toàn không là trạng thái bế tắc
- Trạng thái bế tắc là trạng thái không an toàn
- Trạng thái không an toàn có thể không là trạng thái bế tắc



28

28

Trạng thái an toàn (tt)

- Ví dụ: Xét một hệ thống có 12 tài nguyên là 12 băng từ và 3 tiến trình P0, P1, P2 với các yêu cầu cấp phát:
  - P0 yêu cầu tối đa 10 băng từ
  - P1 yêu cầu tối đa 4 băng từ
  - P2 yêu cầu tối đa 9 băng từ

P	Yêu cầu nhiều nhất	Yêu cầu hiện tại
P0	10	5
P1	4	2
P2	9	7

- Giả sử tại thời điểm t0:
  - P0 đang giữ 5 băng từ
  - P1 và P2: mỗi tiến trình giữ 2 băng từ.
  - → có 3 băng từ rỗi.
- Tại thời điểm t0, hệ thống ở trạng thái an toàn: thứ tự cấp phát <P1, P0, P2> thỏa mãn điều kiện an toàn
- Giả sử ở thời điểm t1, P2 có yêu cầu và được cấp phát 1 băng từ →Hệ thống không ở trạng thái an toàn nữa

Giải thuật đồ thị cấp phát tài nguyên

- Được áp dụng cho các tài nguyên chỉ có 1 thể hiện
- Sử dụng đồ thị cấp phát tài nguyên và bổ sung thêm một cạnh báo trước (claim edge)
- Cạnh báo trước Pi→Rj cho biết Pi có thể yêu cầu cấp phát tài nguyên Rj, được biểu diễn trên đồ thị bằng các đường có nét đứt.
- Các tài nguyên mà tiến trình cần phải được thông báo trước: tất cả các cạnh báo trước của Pi phải xuất hiện trong đồ thị cấp phát tài nguyên.

Giải thuật đồ thị cấp phát tài nguyên (tt)

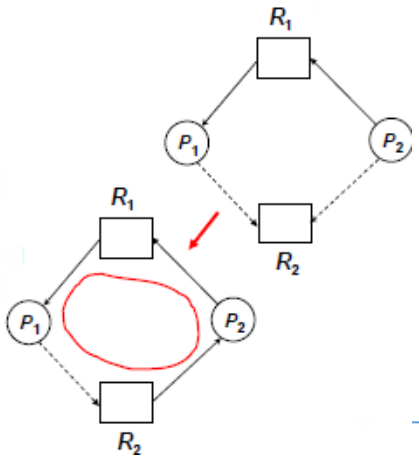
- Giả sử Pi yêu cầu tài nguyên Rj. Yêu cầu này chỉ có thể được chấp nhận nếu ta chuyển cạnh báo trước Pi→Rj thành cạnh cấp phát Rj→Pi mà không tạo ra một chu trình
- Kiểm tra bằng cách sử dụng thuật toán phát hiện chu trình trong đồ thị: Nếu có n tiến trình trong hệ thống, thuật toán phát hiện chu trình có độ phức tạp tính toán O(n2)
  - Nếu không có chu trình: trạng thái an toàn →cấp phát
  - Ngược lại: trạng thái không an toàn.

31

31

Giải thuật đồ thị cấp phát tài nguyên (tt)

- Ví dụ: Giả sử P2 yêu cầu cấp phát R2
- Mặc dù R2 rãnh nhưng chúng ta không thể cấp phát R2, vì nếu cấp phát ta sẽ có chu trình trong đồ thị và gây ra chờ vòng → Hệ thống ở trạng thái không an toàn
- Nhận xét: không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên



32

32

Thuật toán banker

- Còn gọi là giải thuật nhà băng
- Được dùng cho các hệ thống mà mỗi tài nguyên có nhiều thể hiện, kém hiệu quả hơn thuật toán đồ thị phân phối tài nguyên
- Thuật toán banker có thể dùng trong ngân hàng: không bao giờ cấp phát tài nguyên (tiền) gây nên tình huống sau này không đáp ứng được nhu cầu của tất cả các khách hàng

Thuật toán banker (tt)

- Khi một tiến trình mới đưa vào hệ thống, hệ thống phải biết tiến trình cần tối đa bao nhiêu thể hiện của mỗi loại tài nguyên để có thể thực thi.
- Khi tiến trình yêu cầu các tài nguyên, hệ thống phải xác định việc cấp phát các tài nguyên này có đảm bảo hệ thống ở trạng thái an toàn hay không.
  - an toàn: cấp tài nguyên
  - không an toàn: tiến trình phải chờ cho tới khi đủ tài nguyên.

Thuật toán banker (tt)

- Gọi n là số tiến trình trong hệ thống và m là số loại tài nguyên trong hệ thống, cần có các cấu trúc dữ liệu:
  - **Available**: một vector có chiều dài m biểu diễn số lượng tài nguyên sẵn có của mỗi loại.
    - Nếu Available[j]= k → có k thể hiện của loại tài nguyên Rj sẵn dùng.
  - **Max**: ma trận n x m biểu diễn số lượng tối đa yêu cầu của mỗi tiến trình đối với mỗi loại tài nguyên
    - Nếu Max [i , j] = k → Pi có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên Rj.

35

35

Thuật toán banker (tt)

- **Allocation**: ma trận n x m biểu diễn số lượng tài nguyên của mỗi loại mà mỗi tiến trình đang giữ.
  - Nếu Allocation [i, j] = k → Pi hiện được cấp k thể hiện của loại tài nguyên Rj
- **Need**: ma trận n x m biểu diễn số lượng tài nguyên của mỗi loại mà mỗi tiến trình đang cần thêm.
  - Nếu Need [i, j] = k → tiến trình Pi có thể cần thêm k thể hiện của loại tài nguyên Rj để hoàn thành tác vụ của nó.
  - Need [i, j] = Max[i, j] – Allocation [i, j]
- Số lượng và giá trị các biến trên biến đổi theo trạng thái của hệ thống

36

36

Thuật toán banker (tt)

- Qui ước so sánh hai vector: gọi X và Y là các vector có chiều dài n, ta có:
  - X ≤ Y nếu và chỉ nếu X [i] ≤ Y [i] ∀i
  - Thí dụ:
    - X = (1, 7, 3, 2)
    - Y = (0, 3, 2, 1)
    - ➔ Y ≤ X.

37

37

Thuật toán banker (tt)

- **Thuật toán trạng thái an toàn**:
  1. Với Work và Finish là hai vector độ dài m và n. Khởi tạo:
     - Work = Available
     - Finish[i] = false for $i = 0, 1, ..., n - 1$.
  2. Tìm i sao cho Finish [i]==false và Need[i] ≤ Work
     - Nếu không tìm được i, chuyển đến bước 4
  3. Work = Work + Allocation [i],
     Finish [i]=true
     Chuyển đến bước 2
  4. Nếu Finish [i] == true ∀i thì hệ thống ở trạng thái an toàn
- Độ phức tạp tính toán của thuật toán trạng thái an toàn: $O(m.n^2)$

38

38

Thuật toán banker (tt)

- **Thuật toán yêu cầu tài nguyên**:
  1. Nếu Request [i] ≤ Need [i], chuyển đến bước 2
     Ngược lại thông báo lỗi (không có tài nguyên rỗi)
  2. Nếu Request [i] ≤ Available, chuyển đến bước 3.
     Ngược lại Pi phải chờ vì không có tài nguyên
  3. Giả sử hệ thống cấp phát các tài nguyên cho tiến trình Pi → cập nhật trạng thái như sau :
     - Available= Availalble – Request[i]
     - Allocation[i] = Allocation[i] + Request[i]
     - Need[i] = Need [i] - Request[i]
  4. Áp dụng thuật toán kiểm tra trạng thái an toàn:
     - Nếu hệ thống ở trạng thái an toàn → cấp phát tài nguyên cho Pi
     - Ngược lại → Pi phải chờ và trạng thái của hệ thống được khôi phục như cũ

Thuật toán banker (tt)

- Ví dụ: hệ thống gồm
  - 3 loại tài nguyên A, B, C.
    - A có 10 thể hiện
    - B có 5 thể hiện
    - C có 7 thể hiện
  - 5 tiến trình P0, P1, P2, P3, P4
  - Giả sử trạng thái của hệ thống tại thời điểm T0:

1. Tại thời điểm T0, hệ thống có ở trạng thái an toàn hay không
2. Giả sử P1 yêu cầu cấp phát (1,0 2). Yêu cầu này có thể được thỏa mãn hay không?
   Tương tự với:
   P4(3,3,0)
   P0(0,2,0)
   P3(0,2,1)

	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			

Thuật toán banker (tt)

	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			

- Ma trận Need = Max – Allocation

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

- Hệ thống hiện ở trong trạng thái an toàn do thứ tự <P1, P3, P4, P0, P2> thỏa tiêu chuẩn an toàn.

41

41

Thuật toán banker (tt)

- Giả sử rằng quá trình P1 yêu cầu thêm (1,0,2)
  - Request[1](1,0,2) ≤ Available (3,3,2) → True
  - Request[1](1,0,2) <= Need[1] (1,2,2) → True
  - Cập nhật trạng thái mới:

	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	

  - Kiểm tra bằng thuật toán trạng thái an toàn

42

42

- Một hệ thống có 5 tiến trình: P1,…, P5. Trong đó có 3 loại tài nguyên: A (có 8 thực thể), B (có 7 thực thể), C (có 7 thực thể). Sơ đồ cấp phát trong hệ thống tại thời điểm T0 được thể hiện ở bảng sau:

P	Max			Allocation		
	A	B	C	A	B	C
P1	4	3	6	1	1	0
P2	0	4	4	0	2	1
P3	4	2	2	1	1	1
P4	1	6	3	0	0	2
P5	7	3	2	2	1	0

- Áp dụng thuật toán Banker, cho biết hệ thống trên có tồn tại trạng thái an toàn không?

3.3.3 Phát hiện Deadlock

- Nếu không áp dụng phòng tránh hoặc ngăn chặn deadlock thì hệ thống có thể bị deadlock. Khi đó:
  - Cần có thuật toán kiểm tra trạng thái để xem có deadlock xuất hiện hay không
  - Thuật toán khôi phục nếu deadlock xảy ra
- Cần có giải thuật áp dụng cho hệ thống mà mỗi loại tài nguyên chỉ có một thể hiện và hệ thống mà mỗi loại tài nguyên có nhiều thể hiện

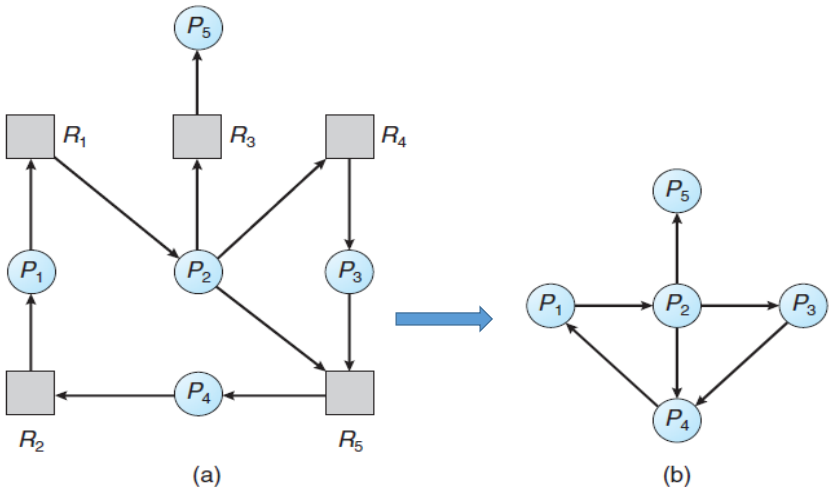
Tài nguyên chỉ có một thể hiện

- **Sử dụng thuật toán đồ thị chờ** (wait-for) có được từ đồ thị cấp phát tài nguyên bằng cách xóa các đỉnh tài nguyên và nối các cung liên quan
- Pi→Pj: Pi đang chờ Pj giải phóng tài nguyên mà Pi cần
- Pi→Pj: tồn tại trong đồ thị chờ nếu và chỉ nếu đồ thị cấp phát tài nguyên tương ứng có hai cung:
  - Pi→Rq
  - Rq→Pj
- Hệ thống có deadlock nếu đồ thị chờ có chu trình
- Để phát hiện deadlock:
  - Cần cập nhật đồ thị chờ
  - Thực hiện định kỳ thuật toán phát hiện chu trình

45

45

Tài nguyên chỉ có một thể hiện (tt)



(a) Resource-allocation graph. (b) Corresponding wait-for graph.

46

46

Tài nguyên có nhiều thể hiện

- Sử dụng các cấu trúc dữ liệu tương tự như giải thuật Banker:
  - **Available**: vector m phần tử biểu diễn số lượng thể hiện của mỗi loại tài nguyên
  - **Allocation**: Ma trận n x m biểu diễn số thể hiện của mỗi loại tài nguyên đang được cấp phát cho các tiến trình
  - **Request**: Ma trận n x m biểu diễn yêu cầu hiện tại của mỗi tiến trình.
    - Nếu Request [i][j] = k → tiến trình Pi yêu cầu cấp phát k thể hiện của tài nguyên Rj

47

47

Tài nguyên có nhiều thể hiện (tt)

1. Khởi tạo:
   - Work = Available
   - For *i* = 0, 1, ..., *n–1*
     - nếu Allocation [i] ≠ 0 gán Finish [i] = false
     - ngược lại gán Finish [i] = true
2. Tìm i sao cho Finish[i]==false và Request[i] ≤ Work. Nếu không tìm thấy i, chuyển đến bước 4
3. Work = Work + Allocation[i], Finish[i] = true → chuyển đến bước 2
4. Nếu Finish[i]==false với 0 ≤ i ≤ n-1 thì hệ thống đang bị bế tắc (và tiến trình Pi đang bế tắc).

- Độ phức tạp tính toán của thuật toán: $O(m.n^2)$

48

48

Tài nguyên có nhiều thể hiện (tt)

- Ví dụ: 5 tiến trình P0, P1, P2, P3, P4
  3 loại tài nguyên A, B, C.
  - A có 7 thể hiện
  - B có 2 thể hiện
  - C có 6 thể hiện.
  - Trạng thái cấp phát tài nguyên tại thời điểm T0:

	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			

  - hệ thống không ở trong trạng thái deadlock (thứ tự<P0, P2, P3, P4, P1> → Finish[i] = true cho với mọi i)

Tài nguyên có nhiều thể hiện (tt)

- Giả sử P2 yêu cầu thêm
  1 thể hiện của tài nguyên C

	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

  - Hệ thống bị deadlock bao gồm các tiến trình P1, P2, P3, P4.

Sử dụng giải thuật phát hiện deadlock

- Khi nào sử dụng giải thuật phát hiện deadlock? Phụ thuộc vào hai yếu tố:
  - Deadlock có khả năng xảy ra thường xuyên?
  - Bao nhiêu tiến trình sẽ bị ảnh hưởng khi có deadlock?
- Sử dụng thuật toán phát hiện:
  - Theo định kỳ: có thể có nhiều chu trình trong đồ thị, không biết được tiến trình/request nào gây ra bế tắc
  - Khi có yêu cầu cấp phát tài nguyên: tốn tài nguyên CPU

51

51

3.3.4 Phục hồi deadlock

1. Kết thúc tiến trình

- Huỷ bỏ tất cả tiến trình bị deadlock:
  - phá vỡ chu trình deadlock
  - chi phí cao
- Hủy bỏ từng tiến trình tại mỗi thời điểm cho đến khi chu trình deadlock bị xóa, tiêu chí chọn để hủy một tiến trình:
  - Độ ưu tiên
  - Thời gian đã thực hiện và thời gian còn lại
  - Số lượng và các loại tài nguyên đã sử dụng
  - Các tài nguyên cần cấp phát thêm
  - Số lượng các tiến trình phải kết thúc
  - Tiến trình là tương tác hay xử lý theo lô (batch)

52

52

Phục hồi deadlock

2. Lấy lại tài nguyên

- Chọn tài nguyên nào và tiến trình nào để thực hiện?
  - Thứ tự ưu tiên.
  - Số lượng tài nguyên mà tiến trình bị deadlock đang giữ
  - Thời gian mà tiến trình đã thực thi
- Khôi phục trạng thái của tiến trình đã chọn như thế nào?
  - Khôi phục tiến trình về trạng thái an toàn và khởi động lại từ trạng thái đó: khó
  - Hủy bỏ tiến trình và khởi động lại để phá vỡ deadlock.
  - Yêu cầu hệ thống phải lưu trữ thông tin về trạng thái của tất cả các tiến trình đang chạy.
- Làm thế nào để tránh tình trạng một tiến trình luôn bị bắt buộc giải phóng tài nguyên?

53