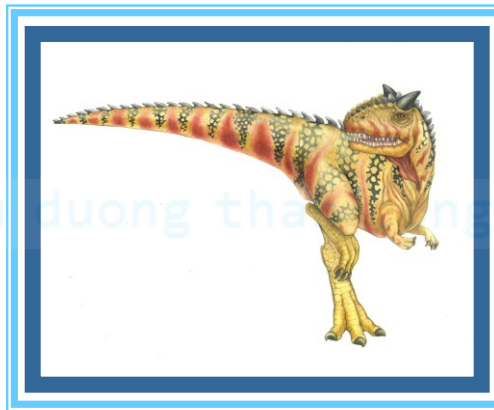


Chương 6: Deadlocks - 2





Câu hỏi ôn tập chương 6 - 1

- Deadlock là gì? Cho ví dụ trong thực tế?
- Một tiến trình khi nào gọi là bị deadlock? trì hoãn vô hạn định?
- Khi nào sẽ xảy ra deadlock?
- Các phương pháp giải quyết deadlock?
- Làm gì để ngăn deadlock?
- Làm gì để tránh deadlock?

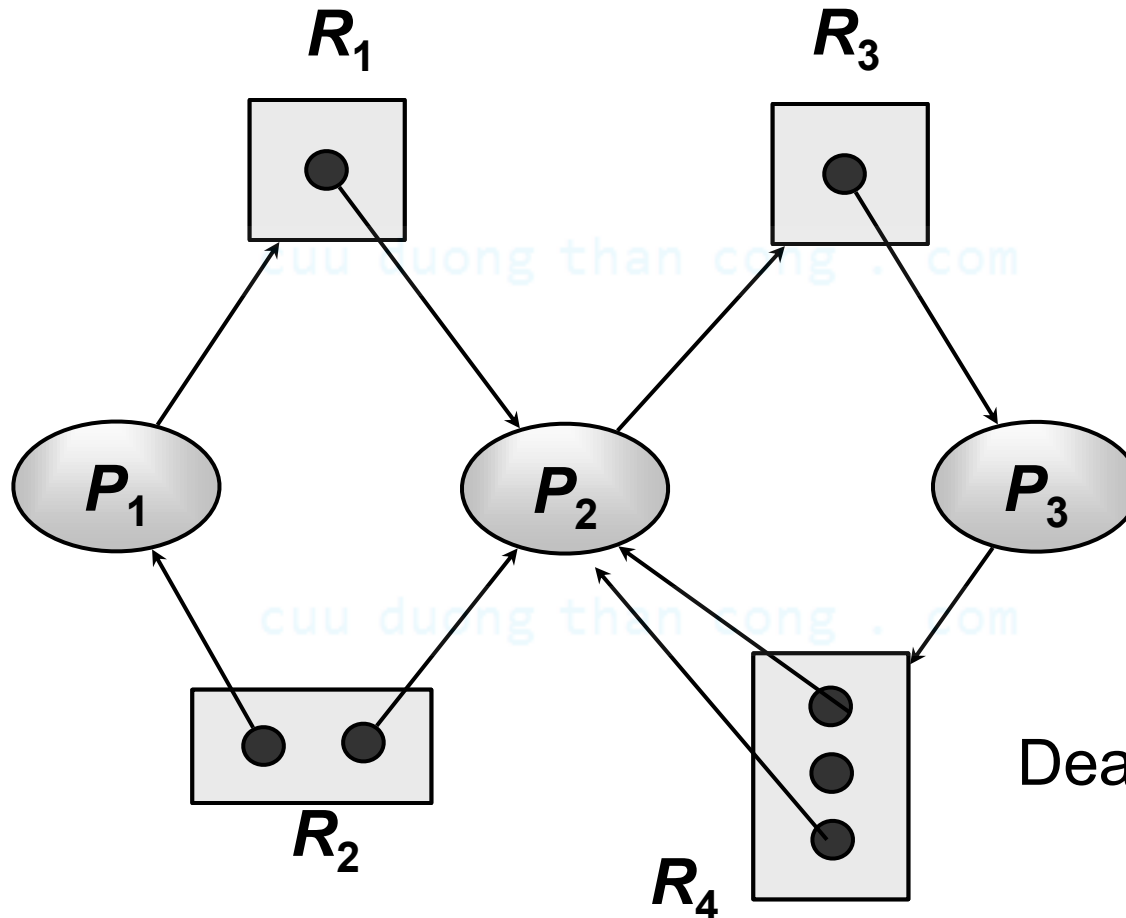




Câu hỏi ôn tập chương 6 – 1 (tt)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

■ Sơ đồ sau có xảy ra deadlock?



Deadlock ?





Câu hỏi ôn tập chương 6 – 1 (tt)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

- Hệ thống có 18 tape drive và 4 tiến trình P0, P1, P2, P3
 - Tại thời điểm t_0

	Max	Allocation	Need	Available
P0	10	5	5	5
P1	4	2	2	3
P2	15	2	13	16
P3	10	6	4	10

cuu duong than cong . com



Deadlocks



Mục tiêu

- Hiểu được thêm các phương pháp giải quyết deadlock
 - Tránh deadlock
 - Phát hiện
 - Phục hồi
- Hiểu và hiện thực được giải thuật Banker





Nội dung

- Giải thuật đồ thị cấp phát tài nguyên
- Giải thuật banker
- Phát hiện deadlock
- Phục hồi deadlock

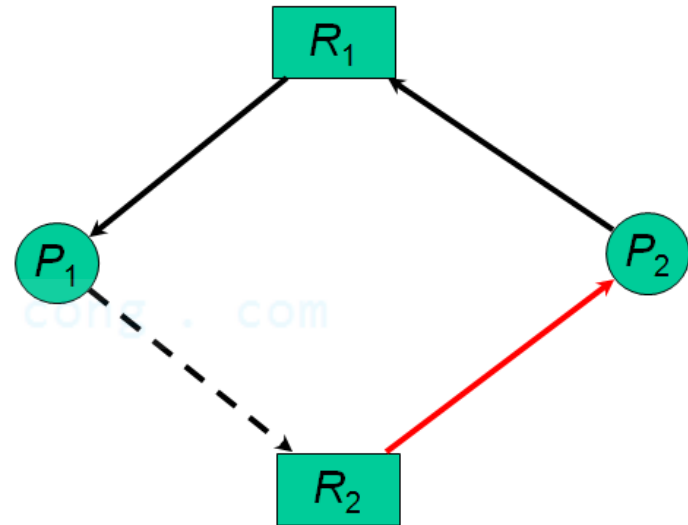
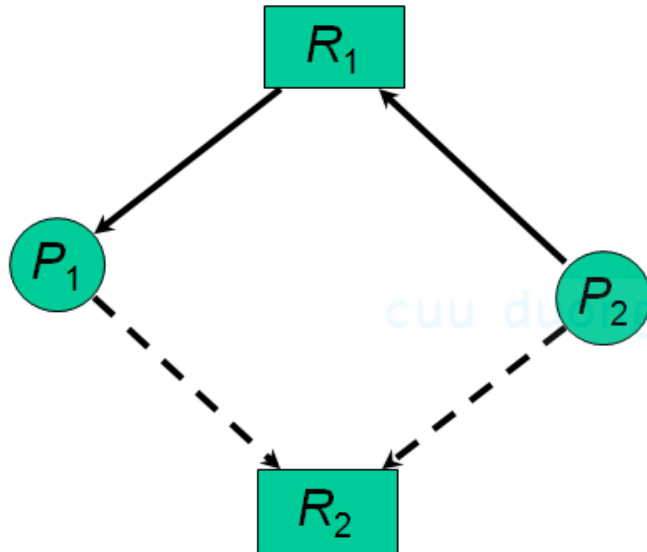
cuu duong than cong . com



Deadlocks



Giải thuật đồ thị cấp phát tài nguyên





Giải thuật Banker

- Mỗi loại tài nguyên có nhiều thực thể
- Bắt chước nghiệp vụ ngân hàng
- Điều kiện:
 - Mỗi tiến trình phải khai báo số lượng thực thể tối đa của mỗi loại tài nguyên mà nó cần
 - Khi tiến trình yêu cầu tài nguyên thì có thể phải đợi
 - Khi tiến trình đã có được đầy đủ tài nguyên thì phải hoàn trả trong một khoảng thời gian hữu hạn nào đó





Cấu trúc dữ liệu cho giải thuật Banker

n : số tiến trình; m : số loại tài nguyên

■ **Available**: vector độ dài m

- $\text{Available}[j] = k \Leftrightarrow$ loại tài nguyên R_j có k instance sẵn sàng

■ **Max**: ma trận $n \times m$

- $\text{Max}[i, j] = k \Leftrightarrow$ tiến trình P_i yêu cầu tối đa k instance của loại tài nguyên R_j

■ **Allocation**: ma trận độ dài $n \times m$

- $\text{Allocation}[i, j] = k \Leftrightarrow P_i$ đã được cấp phát k instance của R_j

■ **Need**: ma trận độ dài $n \times m$

- $\text{Need}[i, j] = k \Leftrightarrow P_i$ cần thêm k instance của R_j

➔ $\text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]$

Ký hiệu $Y \leq X \Leftrightarrow Y[i] \leq X[i]$, với mọi i .

Ví dụ $(0, 3, 2, 1) \leq (1, 7, 3, 2)$





Giải thuật an toàn

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

$$\text{Work} = \text{Available}$$

$$\text{Finish}[i] = \text{false}, i = 0, 1, \dots, n-1$$

2. Tìm i thỏa

(a) $\text{Finish}[i] == \text{false}$

(b) $\text{Need}_i \leq \text{Work}$ (hàng thứ i của Need)

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

3. $\text{Work} = \text{Work} + \text{Allocation}_i$

$$\text{Finish}[i] = \text{true}$$

quay về bước 2

4. Nếu $\text{Finish}[i] == \text{true}, i = 1, \dots, n$, thì hệ thống đang ở trạng thái safe





Giải thuật Banker - Ví dụ

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

- 5 tiến trình P_0, \dots, P_4
- 3 loại tài nguyên:
 - A (10 thực thể), B (5 thực thể), C (7 thực thể)
- 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
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1



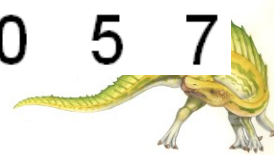


Giải thuật Banker - Ví dụ (tt)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

■ Chuỗi an toàn <P1, P3, P4, P2, P0>

	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





Giải thuật yêu cầu tài nguyên cho tiến trình P_i

$Request_i$ 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 tiến trình 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.
3. Giả định cấp phát tài nguyên đáp ứng yêu cầu của P_i bằng cách cập nhật trạng thái hệ thống như sau:

$$Available = Available - Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$





Giải thuật yêu cầu tài nguyên cho tiến trình P_i (tt)

- Áp dụng giải thuật kiểm tra trạng thái an toàn lên trạng thái trên hệ thống mới
 - 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

$$\text{Available} = \text{Available} + \text{Request}_i$$

$$\text{Allocation}_i = \text{Allocation}_i - \text{Request}_i$$

$$\text{Need}_i = \text{Need}_i + \text{Request}_i$$





Ví dụ: P1 yêu cầu (1, 0, 2)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

■ Kiểm tra Request $1 \leq \text{Available}$:

● $(1, 0, 2) \leq (3, 3, 2) \rightarrow \text{Đúng}$

	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 $\langle P_1, P_3, P_4, P_0, P_2 \rangle$)
vậy có thể cấp phát tài nguyên cho P1



Deadlocks



Ví dụ: P4 yêu cầu (3, 3, 0)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

■ Kiểm tra Request $4 \leq \text{Available}$:

● $(3, 3, 0) \leq (3, 3, 2) \rightarrow \text{Đúng}$

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

■ Trạng thái mới là unsafe vậy không thể cấp phát tài nguyên cho P4





Ví dụ: P0 yêu cầu (0, 2, 0)

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

■ Kiểm tra Request $4 \leq \text{Available}$:

● $(0, 2, 0) \leq (3, 3, 2) \rightarrow \text{Đúng}$

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	3	0	7	2	3	3	1	2
P_1	3	0	2	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			

■ Trạng thái mới là safe, chuỗi an toàn $\langle P_3, P_1, P_2, P_0, P_4 \rangle$
vậy có thể cấp phát tài nguyên cho P0





Phát hiện deadlock

- Chấp nhận xảy ra deadlock trong hệ thống
- Giải thuật phát hiện deadlock
- Cơ chế phục hồi

cuu duong than cong . com

cuu duong than cong . com





Mỗi loại tài nguyên chỉ có một thực thể

■ Sử dụng wait-for graph

- Các Node là các tiến trình
- $P_i \rightarrow P_j$ nếu P_i chờ tài nguyên từ P_j

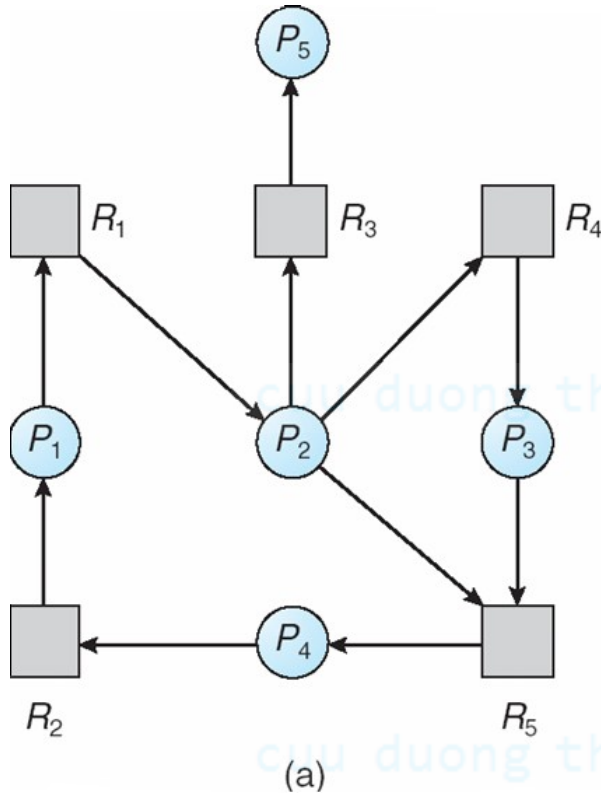
■ Mỗi giải thuật kiểm tra có tồn tại chu trình trong wait-for graph hay không sẽ được gọi định kỳ. Nếu có chu trình thì tồn tại deadlock

■ Giải thuật phát hiện chu trình có thời gian chạy là $O(n^2)$, với n là số đỉnh của graph

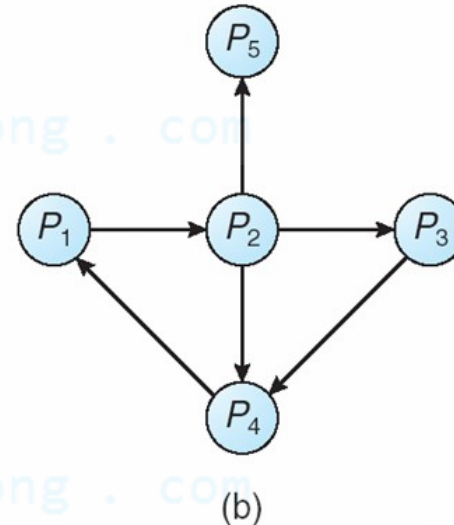




Sơ đồ cấp phát tài nguyên và sơ đồ wait-for



Resource-Allocation Graph



Corresponding wait-for graph





Mỗi loại tài nguyên có nhiều thực thể

- Available: vector độ dài m chỉ số instance sẵn sàng của mỗi loại tài nguyên
- Allocation: ma trận $n \times m$ định nghĩa số instance của mỗi loại tài nguyên đã cấp phát cho mỗi process
- Request: ma trận $n \times m$ chỉ định yêu cầu hiện tại của mỗi tiến trình.
 - Request $[i,j] = k \Leftrightarrow P_i$ đang yêu cầu thêm k instance của R_j





Giải thuật phát hiện deadlock

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

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

- $Work = Available$
- For $i = 0, 1, \dots, n-1$, nếu $Allocation_i \neq 0$ thì $Finish[i] = false$; còn không thì $Finish[i] = true$

2. Tìm i thỏa mãn:

- $Finish[i] == false$
- $Request_i \leq Work$

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

3. $Work = Work + Allocation_i$

$Finish[i] = true$

quay về bước 2.

4. Nếu $Finish[i] == false$, với một số $i = 0, \dots, n-1$, thì hệ thống đang ở trạng thái **deadlock**. Hơn thế nữa, $Finish[i] == false$ thì P_i bị **deadlocked**.

Thời gian chạy của giải thuật $O(m \cdot n^2)$



Deadlocks



Giải thuật phát hiện deadlock - Ví dụ

- 5 quá trình P0 ,..., P4 với 3 loại tài nguyên:
 - A (7 instance), B (2 instance), C (6 instance).
- 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			





Giải thuật phát hiện deadlock - Ví dụ

- 5 quá trình P_0, \dots, P_4 với 3 loại tài nguyên:
 - A (7 instance), B (2 instance), C (6 instance).
- 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			

Chuỗi $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ sẽ cho kết quả $\text{Finish}[i] = \text{true}, i = 1, \dots, n$





Giải thuật phát hiện deadlock - Ví dụ (tt)

- P2 yêu cầu thêm một instance của C. Ma trận Request như sau:

	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	1			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

➔ Hệ thống bị deadlock

(Trong tình trạng này, dù P0 thực thi xong, trả tài nguyên lại cho hệ thống thì lượng tài nguyên này cũng không đủ để đáp ứng Request của các process còn lại ➔ Deadlock)





Phục hồi deadlock

- Khi deadlock xảy ra, để phục hồi
 - Báo người vận hành
 - 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



Deadlocks



Chăm dứt quá trình

■ Chăm dứt quá trình bị deadlock

- Chăm dứt lần lượt từng tiến trình 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 chăm dứt?

- Độ ưu tiên của tiến trình
- Thời gian đã thực thi của tiến trình và thời gian còn lại
- Loại tài nguyên mà tiến trình đã sử dụng
- Tài nguyên mà tiến trình cần thêm để hoàn tất công việc
- Số lượng tiến trình cần được chăm dứt
- Tiến trình là interactive hay batch





Lấy lại tài nguyên

- Lấy lại tài nguyên từ một tiến trình, cấp phát cho tiến trình khác cho đến khi không còn deadlock nữa.
- Chọn “nạn nhân” để tối thiểu chi phí (có thể dựa trên số tài nguyên sở hữu, thời gian CPU đã tiêu tốn,...)
- Trở lại trạng thái trước deadlock (Rollback):
 - Rollback tiến trình bị lấy lại tài nguyên trở về trạng thái safe, tiếp tục tiến trình từ trạng thái đó.
 - Hệ thống cần lưu giữ một số thông tin về trạng thái các tiến trình đang thực thi.
- Đói tài nguyên (Starvation): để tránh starvation, phải bảo đảm không có tiến trình sẽ luôn luôn bị lấy lại tài nguyên mỗi khi deadlock xảy ra.





Phương pháp kết hợp để giải quyết deadlock

■ Kết hợp 3 phương pháp cơ bản

- Ngăn chặn (Prevention)
- Tránh (Avoidance)
- Phát hiện (Detection)

Cho phép sử dụng cách giải quyết tối ưu cho mỗi lớp tài nguyên trong hệ thống.

■ Phân chia tài nguyên thành các lớp theo thứ bậc.

- Sử dụng kỹ thuật thích hợp nhất cho việc quản lý deadlock trong mỗi lớp này.





- Giải thuật đồ thị cấp phát tài nguyên
- Giải thuật banker
- Phát hiện deadlock
- Phục hồi deadlock

cuu duong than cong . com



Deadlocks



Bài tập

- Cho 1 hệ thống có 4 tiến trình P1 đến P4 và 3 loại tài nguyên R1 (3), R2 (2) R3 (2). P1 giữ 1 R1 và yêu cầu 1 R2; P2 giữ 2 R2 và yêu cầu 1 R1 và 1 R3; P3 giữ 1 R1 và yêu cầu 1 R2; P4 giữ 2 R3 và yêu cầu 1 R1
 - Vẽ đồ thị tài nguyên cho hệ thống này?
 - Deadlock?
 - Chuỗi an toàn? (nếu có)





Bài tập

- Tìm Need?
- Hệ thống có an toàn không?
- Nếu P_1 yêu cầu $(0,4,2,0)$ thì có thể cấp phát cho nó ngay không?

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C D	A B C D	A B C D
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	



Kết thúc chương 6

