Họ và tên: Lê Anh Thư

MSSV: 20521985

BÀI TẬP CHƯƠNG 6 HỆ ĐIỀU HÀNH

1. Deadlock là gì?

Trong một môi trường có nhiều process chạy đồng thời, các process có thể tranh giành nhau một số lượng hữu hạn các tài nguyên (resources).

Một process P1 yêu cầu một tài nguyên A, nếu tài nguyên A lúc đó chưa sẵn sàng (available) thì process P1 sẽ vào trạng thái waiting. Đôi khi, process P1 đã vào trạng thái waiting không bao giờ có thể chuyển trạng thái sang Ready nữa, do tài nguyên A đang bị một process P2 nắm giữ, vì P2 cũng đang ở trạng thái waiting đang chờ một tài nguyên B nào đó đang bị nắm giữ cũng bởi >=1 process nào đó (có thể là P1) đang ở trạng thái waiting.

Một trường hợp chờ đợi vòng như vậy, và khi đó hệ thống không thể tiến triển được, người ta gọi là trạng thái **Deadlock**.

Ví dụ :

- Hệ thống có 2 file trên đĩa (A và B).
- P1 đang mở (đang khoá) file A và yêu cầu file B.
- P2 đang mở (đang khoá) file B và yêu cầu file A.

=> Deadlock.

2. Các điều kiện cần để xảy ra deadlock?

- Loại trừ tương hỗ: ít nhất một tài nguyên được giữ theo nonsharable mode

  Ví dụ: printer <> read-only files (sharable)
- Giữ và chờ cấp thêm tài nguyên: Một tiến trình đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do quá trình khác giữ
- Không trưng dụng: tài nguyên không thể bị lấy lại mà chỉ có thể được trả lại từ tiến trình đang giữ tài nguyên đó khi nó muốn
- Chu trình đợi: tồn tại một tập (P0,…,Pn} các quá trình đang đợi sao cho
  - P0 đợi một tài nguyên mà P1 giữ
  - P1 đợi một tài nguyên mà P2 giữ
  - …
  - Pn đợi một tài nguyên mà P0 giữ

3. Đồ thị cấp phát tài nguyên là gì? Mối liên hệ giữa đồ thị cấp phát tài nguyên và deadlock?

Đồ thị cấp phát tài nguyên:

Là đồ thị có hướng, với tập đỉnh V và tập cạnh E

Tập đỉnh V gồm 2 loại:

a. P = {P1, P2,…,Pn} (All process)

b. R = {R1, R2,…,Rn} (All resource)

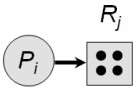
Tập cạnh E gồm 2 loại:

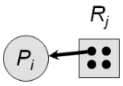
c. Cạnh yêu cầu: Pi -> Rj

d. Cạnh cấp phát: Rj-> Pi

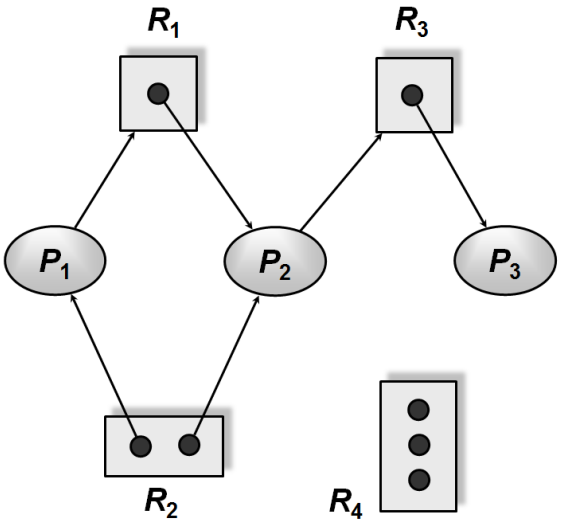
Process i 

Loại tài nguyên Rj với 4 thực thể 

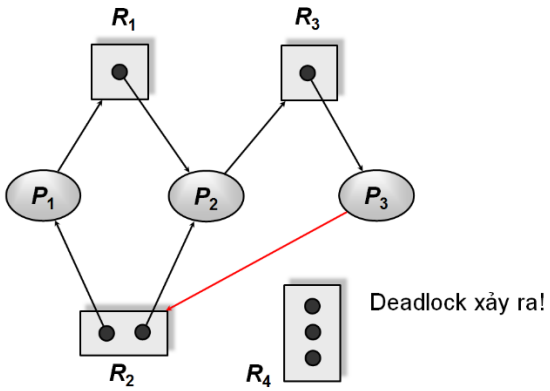
Pi yêu cầu một thực thể của Rj 

Pi đang giữ một thực thể của Rj 

Ví dụ:



Mối liên hệ giữa đồ thị cấp phát tài nguyên và deadlock:



- RAG không chứa chu trình -> 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

4. Có mấy phương pháp để giải quyết deadlock? Phân tích và đánh giá ưu, nhược điểm của từng phương pháp?

❖ Những phương pháp giải quyết deadlock

Nói chung, chúng ta có thể xử lý vấn đề deadlock bằng một trong những cách sau :

- Chúng ta có thể dùng một giao thức để **bảo vệ** hoặc **tránh** deadlock, bảo đảm rằng hệ thống sẽ **không bao giờ** rơi vào trạng thái deadlock.
  - Khác biệt :
    - Ngăn deadlock : Không cho phép ít nhất một trong 4 điều kiện cần cho deadlock xảy ra. Hoạt động bằng cách giới hạn yêu cầu tài nguyên của process.
    - Tránh deadlock : Các process cần cung cấp thông tin về tài nguyên nó cần, để hệ thống cấp phát tài nguyên một cách thích hợp.
- Chúng ta có thể cho phép hệ thống rơi vào trạng thái deadlock, **kiểm tra** deadlock và **phục hồi** hệ thống.
- Chúng ta có thể bỏ qua deadlock và xem như deadlock chưa từng xuất hiện (Windows/Linux). Tuy nhiên, deadlock không được phát hiện, về lâu dài sẽ dẫn đến việc giảm hiệu suất của hệ thống. Cuối cùng, hệ thống có thể ngưng hoạt động và phải khởi động lại.

❖ Phân tích và đánh giá ưu, nhược điểm của từng phương pháp

Vấn đề Deadlock là một trong các vấn đề được nghiên cứu nhiều trong lĩnh vực công nghệ thông tin, các thành tựu trong lĩnh vực đó cho phép đề ra các thuật toán giải quyết nhiều bài toán. Các nghiên cứu có thể chia ra làm 4 hướng chính sau:

- Ngăn chặn Deadlock.
- Tránh Deadlock.
- Phát hiện Deadlock.
- Khôi phục sau Deadlock.

Hướng thứ 1: Ngăn chặn Deadlock: có mục đích tìm những điều kiện để loại trừ khả năng xuất hiện tình trạng Deadlock. Hướng này là giải pháp trực diện đối với vấn đề Deadlock, nhưng nó thường dẫn tới việc sử dụng tài nguyên không hiệu quả. Nhưng dù sao các phương pháp ngăn chặn Deadlock được áp dụng khá phổ biến.

Hướng thứ 2: Tránh Deadlock: Mục đích của các biện pháp tránh Deadlock là ở chỗ cho phép những điều kiện ít nghiêm ngặt hơn so với các biện pháp ngăn chặn Deadlock và cũng đảm bảo sử dụng tài nguyên tốt hơn. Các biện pháp tránh Deadlock không đòi hỏi loại bỏ hoàn toàn để không xảy ra tình trạng deadlock trong hệ thống. Ngược lại nó chú ý các khả năng xuất hiện Deadlock, trong trường hợp xác suất xuất hiện Deadlock tăng lên thì áp dụng các biện pháp tránh xảy ra Deadlock.

Hướng thứ 3: Phát hiện Deadlock: Các phương pháp phát hiện Deadlock áp dụng trong các hệ thống có khả năng xuất hiện Deadlock do hậu quả của các thao tác vô ý hay cố ý. Mục đích của các biện pháp phát hiện là xác định sự tồn tại tình trạng Deadlock trong hệ thống, xác định các tiến trình và tài nguyên nằm trong tình trạng Deadlock. Sau đó có thể áp dụng các biện pháp thích hợp để khắc phục.

Hướng thứ 4: Khôi phục sau Deadlock: Các biện pháp khôi phục sau Deadlock áp dụng khi loại bỏ tình trạng Deadlock để hệ thống có thể tiếp tục hoạt động, còn các tiến trình rơi vào tình trạng Deadlock có thể phải kết thúc và các tài nguyên của nó được giải phóng. Sau đó các tiến trình đó thường được nạp và bắt đầu từ đầu (các công việc đã thực hiện đến lúc xảy ra Deadlock sẽ bị mất).

5. Phân tích và đánh giá ưu, nhược điểm của các giải pháp đồng bộ busy waiting (cả phần cứng và phần mềm)?

- **Các giải pháp phần mềm**

Sử dụng các biến cờ hiệu:

- **Ưu điểm:**

Một giải pháp loại trừ tương hỗ khác không phụ thuộc vào sự hỗ trợ của phần cứng (dưới dạng các lệnh kiểm tra) và tương đối dễ sử dụng.

- **Khuyết điểm:**

Giải pháp này có thể vi phạm điều kiện thứ nhất: hai tiến trình có thể cùng ở trong miền găng tại một thời điểm. Giả sử một tiến trình nhận thấy lock = 0 và chuẩn bị vào miền găng, nhưng trước khi nó có thể đặt lại giá trị cho lock là 1, nó bị tạm dừng để một tiến trình khác hoạt động. Tiến trình thứ hai này thấy lock vẫn là 0 thì vào miền găng và đặt lại lock = 1. Sau đó tiến trình thứ nhất được tái kích hoạt, nó gán lock = 1 lần nữa rồi vaò miền găng. Như vậy tại thời điểm đó cả hai tiến trình đều ở trong miền găng.

Sử dụng việc kiểm tra luân phiên:

- **Ưu điểm:**

Giải pháp này dựa trên việc thực hiện sự kiểm tra nghiêm nhặt đến lượt tiến trình nào được vào miền găng. Do đó nó có thể ngăn chặn được tình trạng hai tiến trình cùng vào miền găng.

- **Nhược điểm:**

Có thể vi phạm điều kiện thứ ba: một tiến trình có thể bị ngăn chặn vào miền găng bởi một tiến trình khác không ở trong miền găng. Giả sử tiến trình B ra khỏi miền găng rất nhanh chóng. Cả hai tiến trình đều ở ngoài miền găng, và turn = 0. Tiến trình A vào miền găng và ra khỏi nhanh chóng, đặt lại giá trị của turn là1, rồi lại xử lý đoạn lệnh ngoài miền găng lần nữa. Sau đó, tiến trình A lại kết thúc nhanh chóng đoạn lệnh ngoài miền găng của nó và muốn vào miền găng một lần nữa. Tuy nhiên lúc này B vẫn còn

mãi xử lý đoạn lệnh ngoài miền găng của mình, và turn lại mang giá trị 1 ! Như vậy, giải pháp này không có giá trị khi có sự khác biệt lớn về tốc độ thực hiện của hai tiến trình, nó vi phạm cả điều kiện thứ hai.

Giải pháp của Peterson:

- **Ưu điểm:**

Giải pháp này ngăn chặn được tình trạng mâu thuẫn truy xuất : mỗi tiến trình Pi chỉ có thể vào miền găng khi *interesse[j]=FALSE* hoặc *turn = i*

- **Nhược điểm:**

Nếu cả hai tiến trình đều muốn vào miền găng thì *interesse[i] = interesse[j] =TRUE* nhưng giá trị của *turn* chỉ có thể hoặc là 0 hoặc là 1, do vậy chỉ có một tiến trình được vào miền găng.

➢ **Các giải pháp phần cứng**

Cấm ngắt

- **Ưu điểm:**

Cho phép tiến trình cấm tất cả các ngắt trước khi vào miền găng, và phục hồi ngắt khi ra khỏi miền găng. Khi đó, ngắt đồng hồ cũng không xảy ra, do vậy hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho tiến trình khác, nhờ đó tiến trình hiện hành yên tâm thao tác trên miền găng mà không sợ bị tiến trình nào khác tranh chấp.

- **Nhược điểm:**

Giải pháp này không được ưa chuộng vì rất thiếu thận trọng khi cho phép tiến trình người dùng được phép thực hiện lệnh cấm ngắt. Hơn nữa, nếu hệ thống có nhiều bộ xử lý, lệnh cấm ngắt chỉ có tác dụng trên bộ xử lý đang xử lý tiến trình, còn các tiến trình hoạt động trên các bộ xử lý khác vẫn có thể truy xuất đến miền găng

Chỉ thị TSL(Test-and-Set)

- **Ưu điểm:**

Chỉ thị TSL giảm nhẹ công việc lập trình để giải quyết vấn đề

- **Nhược điểm:**

Không dễ dàng để cài đặt chỉ thị TSL sao cho được xử lý một cách không thể phân chia, nhất là trên máy với cấu hình nhiều bộ xử lý.

6. Trạng thái an toàn là gì? Mối liên hệ giữa trạng thái an toàn và deadlock?

❖ **Trạng thái an toàn là:**

■ Một trạng thái của hệ thống được gọi là an toàn (safe) nếu tồn tại một chuỗi thứ tự an toàn

■ Một chuỗi quá trình <P1, P2,…,Pn> là một chuỗi an toàn nếu

□ Với mọi i = 1, …, n yêu cầu tối đa về tài nguyên của Pi có thể được thỏa bởi

- Tài nguyên mà hệ thống đang có sẵn sàng
- Cùng với tài nguyên mà tất cả các Pj (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

- Ví dụ: hệ thông có 12 tape drive và 3 tiến trình P0, P1, P2
  - Tại thời điểm t_o

	Cần tối đa	Đang giữ	Cần thêm
P0	10	5	5
P1	4	2	2
P2	9	2	7

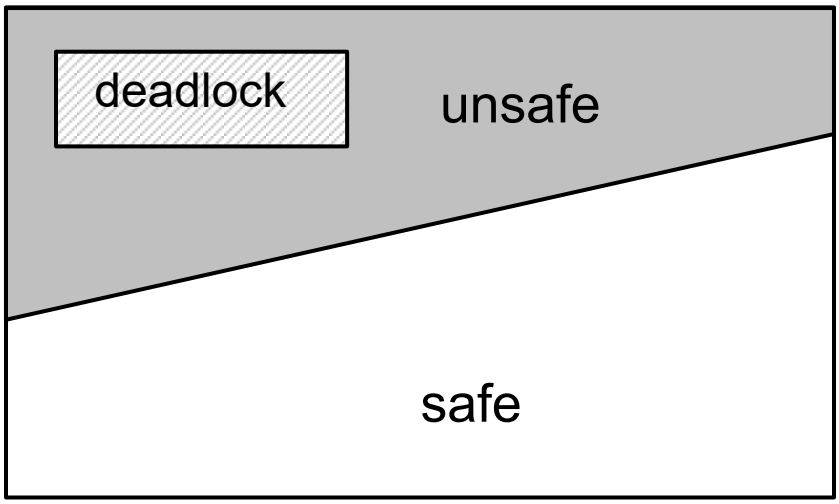
  - Còn 3 tape drive sẵn sàng
  - Chuỗi <P1, P0, P2> là chuỗi an toàn -> hệ thống là an toàn
- Giả sử tại thời điểm t_1, P2 yêu cầu và được cấp phát 1 tape drive
  - Còn 2 tape drive sẵn sàng

	Cần tối đa	Đang giữ
P0	10	5
P1	4	2
P2	9	3

❖ **Mối liên hệ giữa trạng thái an toàn 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



7. Mô tả cách thực hiện các giải thuật Banker: giải thuật an toàn, giải thuật yêu cầu tài nguyên và giải thuật phát hiện deadlock?

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

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

- Available: vector độ dài m
  - Available[j] = k ó loại tài nguyên Rj có k instance sẵn sàng
- Max: ma trận n x m
  - Max[i, j] = k ó tiến trình Pi yêu cầu tối đa k instance của loại tài nguyên Rj
- Allocation: vector độ dài n x m
  - Allocation[i, j] = k ó Pi đã được cấp phát k instance của Rj
- Need: vector độ dài n x m
  - Need[i, j] = k ó Pi cần thêm k instance của Rj
  - => Need[i, j] = Max[i, j] - Allocation[i, j]

Ký hiệu $Y \leq X \Leftrightarrow Y[i] \leq X[i]$, 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 là m và n. Khởi tạo

   Work = Available

   Finish[*i*] = false, *i* = 0, 1, …, *n*-1

2. Tìm *i* thỏa

   (a) Finish[*i*] = false

   (b) $Need_i \leq$ 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

Ví dụ:

- 5 tiến trình P0,…,P4
- 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 T0

	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 yêu cầu tài nguyên**

$Request_i$ 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 – $Request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i – Request_i$

- 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

    Available = Available + $Request_i$

    $Allocation_i = Allocation_i - Request_i$

    $Need_i = Need_i + Request_i$

Ví dụ: P1 yêu cầu (1, 0, 2)

- Kiểm tra Request 1 ≤ Available:
  - (1, 0, 2) ≤ (3, 3, 2) => Đúng
- 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			

- Trạng thái mới là safe (chuỗi an toàn là <P1, P3, P4, P0, P2> vậy có thể cấp phát tài nguyên cho P1

❖ **Giải thuật phát hiện deadlock**

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

   a. Work = Available

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

còn không thì Finish[i] := true

2. Tìm i thỏa mãn:

a. Finish[i] = false

b. $Request_i \leq Work$

- 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 $i = 1,\ldots, n$, 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 $O(m \cdot n^2)$

Ví dụ:

- 5 quá trình P0 ,…, P4 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			

Chuỗi $<P_0, P_2, P_3, P_1, P_4>$ sẽ cho kết quả Finish[i] = true, $i = 1,\ldots, n$

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**			

8. <u>Nêu các giải pháp để phục hồi hệ thống sau khi pháp hiện có deadlock</u>

Các giải pháp để phục hồi hệ thống sau khi pháp hiện có deadlock:

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

Giải thuật phát hiện deadlock:

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

   a. Work = Available

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

   còn không thì Finish[*i*] := true

2. Tìm *i* thỏa mãn:

   a. Finish[*i*] = false

   b. $Request_i \leq Work$

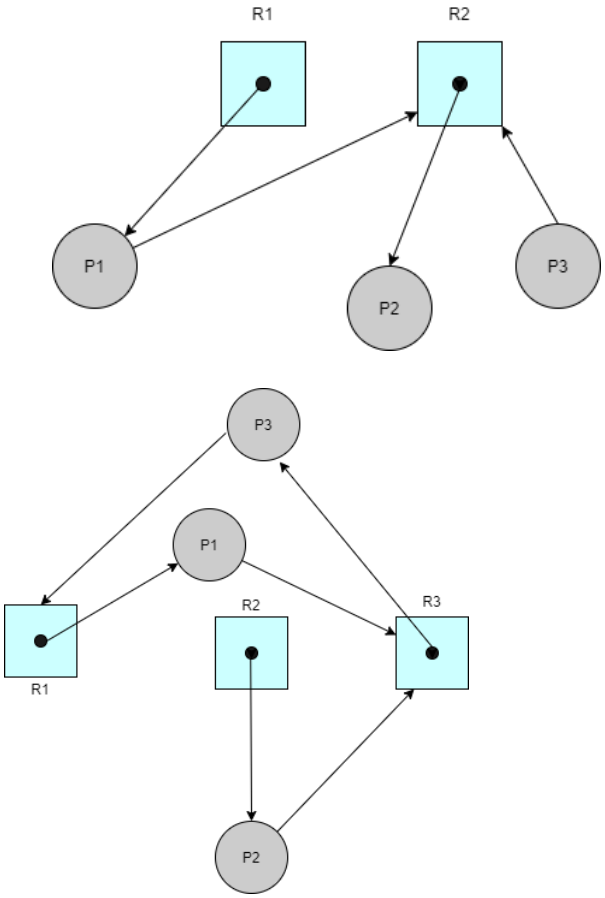
- Nếu không tồn tại *i* như vậy, đến bước 4.

3. $Work = Work + Allocation_i$
Finish[*i*] = true
quay về bước 2.

4. Nếu Finish[*i*] = false, với một số *i* = 1,…, *n*, 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 $O(m \cdot n^2)$

9. <u>(Bài tập mẫu) Cho các đồ thị cấp phát tài nguyên sau. Hỏi đồ thị nào có deadlock xảy ra?</u>



(a) (b)

<u>Trả lời:</u>

- Đồ thị (a) không có deadlock, do đồ thị này có chuỗi an toàn là: <P2, P1, P3> hoặc <P2, P3, P1>.
- Đồ thị (b) có deadlock.

10. <u>(Bài tập mẫu) Cho 1 hệ thống có 4 tiến trình P1, P2, P3, 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.</u>

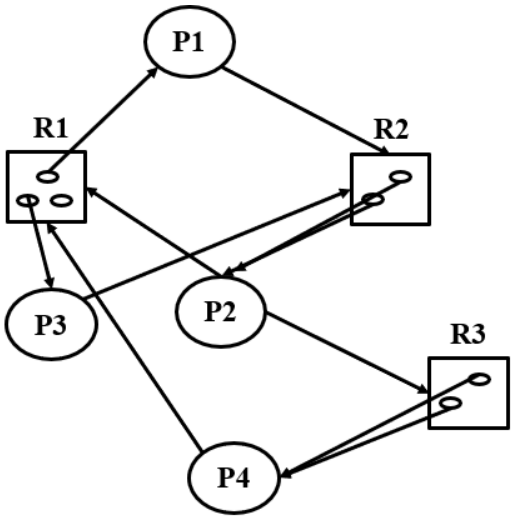
a. Vẽ đồ thị cấp phát tài nguyên của hệ thống

b. Hệ thống có deadlock không?

c. Tìm chuỗi an toàn (nếu có)

<u>Trả lời:</u>

a. Đồ thị cấp phát tài nguyên



b. Hệ thống không bị deadlock do hệ thống có chuỗi an toàn.

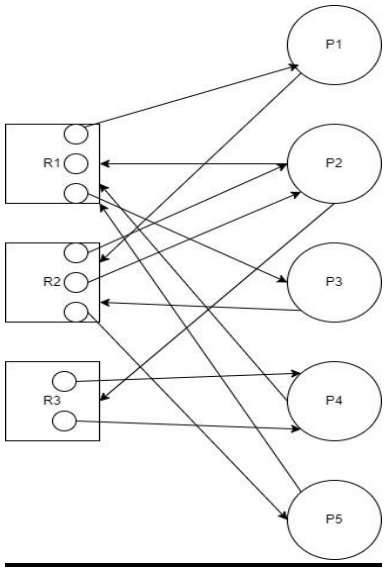
c. Chuỗi an toàn là: <P4, P2, P3, P1> hoặc <P4, P2, P1, P3>.

11. <u>Cho 1 hệ thống có 5 tiến trình P1, P2, P3, P4, P5 và 3 loại tài nguyên R1 (có 3 thực thể), R2 (có 3 thực thể) R3 (có 2 thực thể). P1 giữ 1 thực thể R1 và yêu cầu 1 thực thể R2; P2 giữ 2 thực thể R2 và yêu cầu 1 thực thể R1 và 1 thực thể R3; P3 giữ 1 thực thể R1 và yêu cầu 1 thực thể R2; P4 giữ 2 thực thể R3 và yêu cầu 1 thực thể R1; P5 đang giữ 1 thực thể của R2 và yêu cầu 1 thự thể của R1.</u>

a. Vẽ đồ thị cấp phát tài nguyên
b. Có bao nhiêu chuỗi an toàn cho hệ thống trên?
c. Viết ra tất cả các chuỗi an toàn (nếu có)

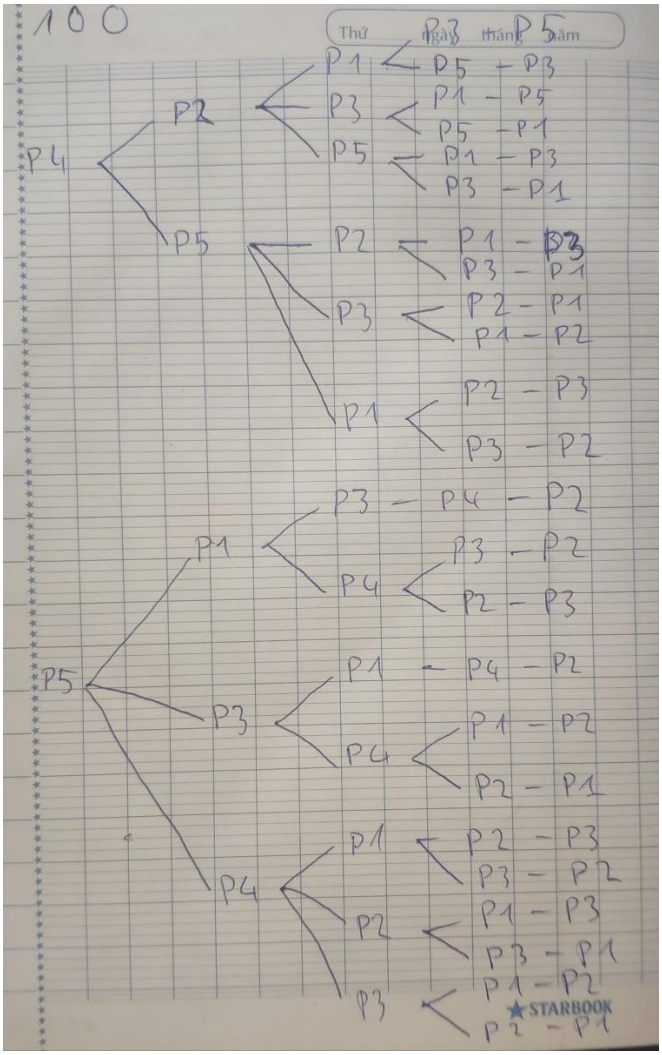
<u>Trả lời:</u>

a. Đồ thị cấp phát tài nguyên



b. Hệ thống có 24 chuỗi an toàn

c. Các chuỗi an toàn:



12. <u>(Bài tập mẫu) Xét một hệ thống máy tính có 5 tiến trình: P0, P1, P2, P3, P4 và 4 loại tài nguyên: A, B, C, D. Tại thời điểm t0, trạng thái của hệ thống như sau:</u>

	Available			
	A	**B**	**C**	**D**
	1	5	2	0

	Allocation				Max			
Tiến trình	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2
P1	1	0	0	0	1	7	5	0
P2	1	3	5	4	2	3	5	6
P3	0	6	3	2	0	6	5	2
P4	0	0	1	4	0	6	5	6

a. Tìm Need?

b. Hệ thống có an toàn không?

c. Nếu P1 yêu cầu (0,4,2,0) thì có thể cấp phát cho nó ngay không?

Trả lời:

a. Ma trận Need

	Need			
Tiến trình	**A**	**B**	**C**	**D**
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

b. Thực hiện giải thuật an toàn

	Allocation				Max				Need				Available (Work)				
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	
P0	0	0	1	2	0	0	1	2	0	0	0	0	1	5	2	0	P0
P1	1	0	0	0	1	7	5	0	0	7	5	0	1	5	3	2	P2
P2	1	3	5	4	2	3	5	6	1	0	0	2	2	8	8	6	P3
P3	0	6	3	2	0	6	5	2	0	0	2	0	2	14	11	8	P4
P4	0	0	1	4	0	6	5	6	0	6	4	2	2	14	12	12	P1

Hệ thống có chuỗi an toàn <P0, P2, P3, P4, P1> cho nên hệ thống an toàn.

c.

Request P1 (0,4,2,0) ≤ Need P1 (0, 7, 5, 0).

Request P1 (0,4,2,0) ≤ Available (1, 5, 2, 0).

Giả sử hệ thống đáp ứng yêu cầu (0,4,2,0) của P1.

Trạng thái mới của hệ thống:

	Allocation				Max				Need				Available (Work)				
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**	
P0	0	0	1	2	0	0	1	2	0	0	0	0	1	1	0	0	P0
P1	1	4	2	0	1	7	5	0	0	3	3	0	1	1	1	2	P2
P2	1	3	5	4	2	3	5	6	1	0	0	2	2	4	6	6	P3
P3	0	6	3	2	0	6	5	2	0	0	2	0	2	10	9	8	P4
P4	0	0	1	4	0	6	5	6	0	6	4	2	2	10	7	12	P1

Hệ thống mới vẫn có chuỗi an toàn <P0, P2, P3, P4, P1> cho nên hệ thống đáp ứng yêu cầu cấp phát cho P1.

13. Xét hệ thống tại thời điểm t_0 có 5 tiến trình: P1, P2, P3, P4, P5; và 4 loại tài nguyên: R1, R2, R3, R4. Xét trạng thái hệ thống như sau:

	Allocation	Max

Available			
R1	R2	R3	R4
2	1	2	0

Process	R1	R2	R3	R4	R1	R2	R3	R4
P1	0	0	1	2	0	0	3	2
P2	2	0	0	0	2	7	5	0
P3	0	0	3	4	6	6	5	6
P4	2	3	5	4	3	3	5	6
P5	0	3	3	2	0	6	5	2

a. Tại thời điểm t_0, áp dụng giải thuật banker tìm chuỗi an toàn của hệ thống?

b. Tại thời điểm t1, tiến trình P3 yêu cầu thêm tài nguyên (1, 1, 0, 0) thì hệ thống có thể đáp ứng ngay được không? Tại sao?

Trả lời:

a)

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	3	2	0	0	2	0	2	1	2	0
P2	2	0	0	0	2	7	5	0	0	7	5	0	2	1	3	2
P3	0	0	3	4	6	6	5	6	6	6	2	2	4	4	8	6
P4	2	3	5	4	3	3	5	6	1	0	0	2	4	7	11	8
P5	0	3	3	2	0	6	5	2	0	3	2	0	6	7	11	8

Chuỗi an toàn: P1 => P4 => P5 => P2 => P3

=> Hệ thống an toàn

b)

Request3 = (1, 1, 0, 0)

Ta có:

Request3 < Need3

Request3 < Available

Giả sử cấp phát tài nguyên P3 thành công:

Allocation3 = Allocation3 + Request3 = (1, 1, 3, 4)

Need3 = Need3 – Request3 = (5, 5, 2, 2)

Available = Available – Request3 = (1, 0, 2, 0)

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	3	2	0	0	2	0	1	0	2	0

P2	2	0	0	0	2	7	5	0	0	7	5	0	1	0	3	2
P3	1	1	3	4	6	6	5	6	5	5	2	2	3	3	8	6
P4	2	3	5	4	3	3	5	6	1	0	0	2	3	6	11	8
P5	0	3	3	2	0	6	5	2	0	3	2	0				

Không tìm được Needi nhỏ hơn Available (3, 6, 11, 8)

Hệ thống không an toàn

Hệ thống không đáp ứng yêu cầu cấp phát cho P3

14. <u>Sử dụng giải thuật Banker để kiểm tra các trạng thái sau có an toàn hay không? Nếu có thì đưa ra chuỗi thực thi an toàn, nếu không thì nêu rõ lý do không an toàn?</u>

a. Available = (0,3,0,1)

b. Available = (1,0,0,2)

	Allocation				**Max**			
Tiến trình	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	3	0	1	4	5	1	1	7
P1	2	2	1	0	3	2	1	1
P2	3	1	2	1	3	3	2	1
P3	0	5	1	0	4	6	1	2
P4	4	2	1	2	6	3	2	5

<u>Trả lời</u>

a) Available = (0,3,0,1)

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	3	0	1	4	5	1	1	7	2	1	0	3	0	3	0	1
P1	2	2	1	0	3	2	1	1	1	0	0	1	3	4	2	2
P2	3	1	2	1	3	3	2	1	0	2	0	0	5	6	3	2
P3	0	5	1	0	4	6	1	2	4	1	0	2	5	11	4	2
P4	4	2	1	2	6	3	2	5	2	1	1	3				

Đáp án câu a.

=> Trạng thái không an toàn do không tìm được giá trị Needi nhỏ hơn Available = (5,11,4,2)

b) Available = (1,0,0,2)

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	3	0	1	4	5	1	1	7	2	1	0	3	1	0	0	2
P1	2	2	1	0	3	2	1	1	1	0	0	1	3	2	1	2
P2	3	1	2	1	3	3	2	1	0	2	0	0	6	3	3	3
P3	0	5	1	0	4	6	1	2	4	1	0	2	6	8	4	3
P4	4	2	1	2	6	3	2	5	2	1	1	3	10	10	5	5

Đáp án câu b.

=> Chuỗi an toàn: P1 -> P2 -> P3 -> P4 -> P0

=> Trạng thái an toàn

15. <u>Trả lời các câu hỏi sau bằng cách sử dụng giải thuật Banker:</u>

a. Hệ thống có an toàn không? Đưa ra chuỗi an toàn nếu có?

b. Nếu P1 yêu cầu (1,1,0,0) thì có thể cấp phát cho nó ngay không?

c. Nếu P4 yêu cầu (0,0,2,0) thì có thể cấp phát cho nó ngay không?

Available			
A	**B**	**C**	**D**
3	3	2	1

	Allocation				**Max**			
Tiến trình	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	2	0	0	1	4	2	1	2
P1	3	1	2	1	5	2	5	2
P2	2	1	0	3	2	3	1	6
P3	1	3	1	2	1	4	2	4
P4	1	4	3	2	3	6	6	5

<u>Trả lời</u>

a)

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	2	2	1	1	3	3	2	1
P1	3	1	2	1	5	2	5	2	2	1	3	1	5	3	2	2
P2	2	1	0	3	2	3	1	6	0	2	1	3	6	6	3	4
P3	1	3	1	2	1	4	2	4	0	2	1	2	7	10	6	6
P4	1	4	3	2	3	6	6	5	2	2	3	3	10	11	8	7

Chuỗi an toàn: P0 -> P3 -> P4 -> P1 -> P2

b) Nếu P1 yêu cầu (1,1,0,0)

Ta có: Request1 <= Need1

Request1 <= Available

Giả sử cấp phát tài nguyên cho P1 thành công:

Available = Available – Request1 = (2,2,2,1)

Need1 = Need1 – Request1 = (1,0,3,1)

Allocation1 = Allocation1 + Request1 = (4,2,2,1)

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	2	2	1	1	2	2	2	1
P1	4	2	2	1	5	2	5	2	1	0	3	1	4	2	2	2
P2	2	1	0	3	2	3	1	6	0	2	1	3	5	5	3	4
P3	1	3	1	2	1	4	2	4	0	2	1	2	6	9	6	6
P4	1	4	3	2	3	6	6	5	2	2	3	3	10	11	8	7

Chuỗi an toàn: P0 -> P3 -> P4 -> P1 -> P2

c) Nếu P4 yêu cầu (0,0,2,0)

Request4 = (0, 0, 2, 0)

Ta có:

Request4 < Need4

Request4 < Available

Giả sử cấp phát tài nguyên P4 thành công:

Allocation4 = Allocation4 + Request4 = (1, 4, 5, 2)

Need4 = Need4 – Request4 = (2, 2, 1, 3)

Available = Available – Request4 = (3, 3, 0, 1)

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	2	2	1	1	3	3	0	1
P1	3	1	2	1	5	2	5	2	2	1	3	1				
P2	2	1	0	3	2	3	1	6	0	2	1	3				
P3	1	3	1	2	1	4	2	4	0	1	1	2				
P4	1	4	5	2	3	6	6	5	2	2	1	3				

Không tìm được Needi nhỏ hơn Available (3, 3, 0, 1)

- ⇨ Hệ thống không an toàn
- ⇨ Hệ thống không đáp ứng yêu cầu cấp phát cho P4