MÔN HỌC: HỆ ĐIỀU HÀNH

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 6

1. Deadlock là gì?

Deadlock là một trạng thái trong hệ thống máy tính khi một nhóm các tiến trình hoặc luồng không thể tiếp tục thực hiện vì mỗi tiến trình trong nhóm đang chờ đợi một tài nguyên mà chỉ có sẵn từ một tiến trình khác trong nhóm. Điều này dẫn đến một tình trạng "đứng yên" vô hạn, khi không có tiến triển nào có thể được thực hiện.

2. Các điều kiện cần để xảy ra deadlock?

Một tình huống deadlock chỉ xảy ra khi có 4 tình huống sau đây cùng xảy ra đồng thời :

- Loại trừ tương hỗ (Mutual Exclusion) : Có ít nhất 1 tài nguyên đang bị chiếm giữ theo cơ chế không chia sẻ (nonshareable mode). Hay nói cách khác, chỉ có một process P1 duy nhất được sử dụng tài nguyên A trong một thời điểm. Nếu một process P2 khác yêu cầu tài nguyên nói trên, process P2 phải chờ đến khi tài nguyên A đã được process P1 giải phóng.
- Giữ và chờ (Hold and Wait) : Một process P1 phải đang giữ ít nhất một tài nguyên A và đang chờ một hoặc nhiều tài nguyên B đang bị giữ bởi một process B2 khác.
- Không trưng dụng (Non-Preemption) : Tài nguyên không thể bị lấy lại, tức là tài nguyên chỉ có thể được giải phóng khi process đang giữ nó trả lại sau khi đã hoàn thành xong công việc.
- Chu trình đợi (Circular Wait) : Tồn tại một tập {P0,…,Pn} các process 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 (Resource Allocation Graph) là một biểu đồ được sử dụng để mô tả quá trình cấp phát và giải phóng tài nguyên trong hệ thống, đặc biệt là trong ngữ cảnh quản lý tài nguyên để tránh deadlock. Đồ thị này thường được sử dụng trong giải thuật tránh deadlock.

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

1. Đỉnh (Vertex): Mỗi tiến trình được biểu diễn bởi một đỉnh trên đồ thị.
2. Cạnh (Edge): Mỗi cạnh trên đồ thị thể hiện một mối quan hệ cấp phát giữa một tiến trình và một tài nguyên. Nếu có cạnh giữa một tiến trình và một tài nguyên, có nghĩa là tiến trình đó đang giữ tài nguyên đó.
3. Quan hệ cấp phát (Allocation Relationship): Nếu một tiến trình yêu cầu một tài nguyên, có một cạnh hướng từ tiến trình đó đến tài nguyên đó trên đồ thị.
4. Quan hệ giải phóng (Request Relationship): Nếu một tiến trình giải phóng một tài nguyên, cạnh giữa tiến trình và tài nguyên đó được loại bỏ.

Deadlock có thể xảy ra khi và chỉ khi có chu trình trong đồ thị cấp phát tài nguyên. Một chu trình đồng nghĩa với việc mỗi tiến trình trong chu trình đang giữ một tài nguyên và đợi một tài nguyên khác được giữ bởi tiến trình khác trong chu trình. Khi một chu trình được phát hiện trong đồ thị cấp phát tài

nguyên, hệ thống có thể xác định được một số tiến trình và tài nguyên có thể dẫn đến deadlock, và sau đó áp dụng các chiến lược như giải phóng một số tài nguyên hoặc kill một số tiến trình để tránh hoặc giải quyết deadlock.

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

Có ba phương pháp chính để giải quyết vấn đề deadlock trong hệ thống máy tính:

1. Ngăn ngừa Deadlock (Deadlock Prevention): Phương pháp này nhằm thiết kế hệ thống sao cho loại trừ khả năng xảy ra deadlock bằng cách ngăn chặn ít nhất một trong bốn điều kiện cần thiết cho deadlock: loại trừ lẫn nhau, giữ và chờ, không có trưng dụng, và chờ đợi vòng tròn.

- Ưu điểm: Đảm bảo hệ thống không bao giờ rơi vào trạng thái deadlock.
- Nhược điểm: Có thể gây lãng phí tài nguyên do các quy trình không được phép giữ và chờ tài nguyên.

2. Tránh Deadlock (Deadlock Avoidance): Sử dụng thuật toán như thuật toán Banker để đánh giá trước các yêu cầu tài nguyên và chỉ cấp phát khi đảm bảo hệ thống ở trạng thái an toàn[1].

- Ưu điểm: Cho phép sử dụng tài nguyên một cách linh hoạt hơn so với phương pháp ngăn ngừa.
- Nhược điểm: Cần thông tin chi tiết về số lượng tài nguyên tối đa mà mỗi quá trình có thể yêu cầu, điều này không phải lúc nào cũng khả thi.

3. Phát hiện và Khôi phục Deadlock (Deadlock Detection and Recovery): Cho phép deadlock xảy ra, sau đó sử dụng thuật toán để phát hiện và giải quyết, thường là bằng cách giết chết một hoặc nhiều quá trình để giải phóng tài nguyên.

- Ưu điểm: Đơn giản hơn trong việc triển khai so với hai phương pháp trên.
- Nhược điểm: Có thể gây mất mát dữ liệu nếu quá trình bị giết chết đang giữ dữ liệu quan trọng.

Ngoài ra, còn có các phương pháp khác như Thuật toán bầu chọn (Voting Algorithm) và Thuật toán gắn nhãn thời gian (Time Stamping Algorithm) thường được sử dụng trong hệ thống phân tán để giải quyết deadlock. Mỗi phương pháp có cách tiếp cận và cơ chế hoạt động khác nhau, tùy thuộc vào môi trường và yêu cầu cụ thể của hệ thống. Đánh giá ưu nhược điểm của từng phương pháp giúp lựa chọn giải pháp phù hợp nhất cho từng tình huống cụ thể.

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

Busy waiting, hay còn gọi là spinlock, là một kỹ thuật đồng bộ hóa trong đó một tiến trình hoặc luồng liên tục kiểm tra một điều kiện cho đến khi nó trở nên đúng. Dưới đây là phân tích và đánh giá ưu nhược điểm của busy waiting:

Ưu điểm:
- Hiệu suất cao trong môi trường đa luồng: Khi thời gian chờ đợi ngắn, busy waiting có thể nhanh chóng phản hồi khi tài nguyên trở nên khả dụng[1].
- Đơn giản trong triển khai: Đối với các tình huống đơn giản, việc sử dụng busy waiting không đòi hỏi cơ chế phức tạp[2].

Nhược điểm:
- Lãng phí CPU: Busy waiting sử dụng CPU để liên tục kiểm tra điều kiện, dẫn đến lãng phí tài nguyên khi CPU có thể được sử dụng cho các tác vụ khác[1].
- Không hiệu quả trong môi trường đa tiến trình: Trong môi trường có nhiều tiến trình cạnh tranh, busy waiting có thể

gây ra hiện tượng starvation, nơi một tiến trình không bao giờ nhận được tài nguyên do các tiến trình khác liên tục chiếm giữ CPU[1].

Trong phần cứng, busy waiting thường được sử dụng với các cấu trúc như semaphore hoặc lock, nơi một tiến trình sẽ liên tục kiểm tra trạng thái của semaphore hoặc lock cho đến khi nó được giải phóng. Trong phần mềm, busy waiting có thể được triển khai thông qua các vòng lặp kiểm tra điều kiện hoặc sử dụng các biến cờ.

Tùy thuộc vào yêu cầu cụ thể của hệ thống và môi trường hoạt động, việc lựa chọn sử dụng busy waiting hay các giải pháp đồng bộ khác như sleep and wakeup cần được cân nhắc kỹ lưỡng để đảm bảo hiệu suất và sử dụng tài nguyên một cách hiệu quả.

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

Trạng thái an toàn trong hệ thống máy tính là trạng thái mà ở đó, mặc dù các tiến trình có thể đang chờ đợi tài nguyên, nhưng vẫn có một lộ trình cấp phát tài nguyên không dẫn đến deadlock. Nói cách khác, có thể tồn tại một chuỗi cấp phát tài nguyên sao cho mọi tiến trình đều có thể hoàn thành mà không bị chặn vĩnh viễn.

Mối liên hệ giữa trạng thái an toàn và deadlock là:
- Trạng thái an toàn đảm bảo rằng hệ thống có thể tiếp tục hoạt động mà không rơi vào trạng thái deadlock.
- Deadlock xảy ra khi hệ thống không còn ở trạng thái an toàn và các tiến trình không thể tiếp tục hoạt động do chờ đợi lẫn nhau vô hạn.
Thuật toán Banker là một ví dụ của phương pháp tránh deadlock, nó sử dụng khái niệm trạng thái an toàn để quyết định liệu có

nên cấp phát tài nguyên cho một tiến trình hay không. Nếu việc cấp phát làm hệ thống rơi vào trạng thái không an toàn, thuật toán sẽ từ chối yêu cầu để tránh deadlock.

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?**

Thuật toán sắp xếp việc cấp phát tài nguyên dựa vào đồ thị RAG không thể áp dụng được đối với các hệ thống có nhiều thực thể (instance) của cùng một loại tài nguyên (ví dụ R có 2 thực thể), lúc đó ta phải dùng đến một giải thuật tránh deadlock có tên là giải thuật Banker. Được gọi là giải thuật Banker là vì thuật toán có thể được sử dụng trong các hệ thống ngân hàng để đảm bảo rằng ngân hàng sẽ không chi tiền đến mức mà nó không thể đáp ứng được nhu cầu của tất cả các khách hàng.

Trạng thái an toàn là trạng thái mà số lượng tài nguyên còn lại (Available) của hệ thống có thể đáp ứng được tất cả các process và tạo ra 1 chuỗi an toàn.

Chuỗi an toàn là một thứ tự thực hiện việc cung cấp tài nguyên cho các process, với giả định rằng các process sẽ trả lại tài nguyên mà nó đang giữ sau khi thực hiện xong.

Hoạt động :

Khi một tiến trình vào hệ thống, tiến trình đó cần phải đưa ra số lượng thực thể tối đa mà nó cần (để từ đó hệ thống có thể xác định trạng thái an toàn, dựa vào việc thoả mãn yêu cầu của tất cả process). Con số này không được vượt quá số lượng tài nguyên hiện đang có trong hệ thống.

Khi một process yêu cầu tài nguyên, hệ thống sẽ xem xét : Liệu sau khi cấp tài nguyên thì hệ thống có còn đang ở trạng thái an toàn hay không ? Nếu có thì hệ thống sẽ cấp, ngược lại hệ thống sẽ không cấp tài nguyên cho process. Lúc đó process phải chờ đến khi các process khác giải phóng đủ số lượng tài nguyên cần thiết.

Cấu trúc dữ liệu cho giải thuật Banker :

Gọi m là số loại tài nguyên đang xét trong giải thuật Banker.

Gọi n là số tiến trình đang xét trong giải thuật Banker.

- Mảng Available có m phần tử : Từng phần tử Available[j] chứa một con số tương ứng với số thể hiện của loại tài nguyên j đó.
- Mảng Max[i,j] = k : Tiến trình Pi yêu cầu tối đa k thực thể của loại tài nguyên Rj.
- Allocation[i,j] = k : Pi đã được cấp phát k thực thể của Rj.
- Need[i,j] = k : Pi cần thêm k thực thể của Rj.
- Need[i,j] = Max[i,j] – Allocation[i,j].

Các bước thực hiện giải thuật :

(Lưu ý : Ký hiệu Y <= X ó Y[i] <= X[i], ví dụ (0,3,2,1) <= (1,7,3,2)).

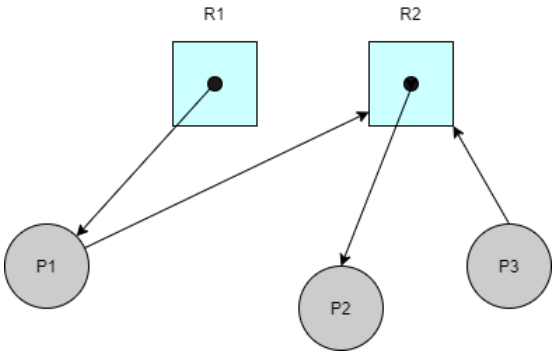
1. Gọi Work và Finish là hai vector độ dài là m và n. Khởi tạo
   Work = Available
   Finish[i] = false, i = 0, 1, …, n-1
2. Tìm i thỏa
   (a) Finish[i] = false
   (b) $Need_i \leq$ Work (hàng thứ i của Need)
   Nếu không tồn tại i như vậy, đến bước 4.
3. Work = Work + $Allocation_i$
   Finish[i] = true
   quay về bước 2
4. Nếu Finish[i] = true, i = 1,…, n, thì hệ thống đang ở trạng thái safe

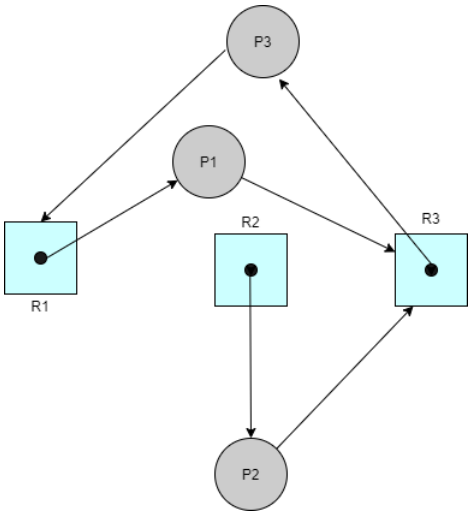
8. Nêu các giải pháp để phục hồi hệ thống sau khi phát hiện có deadlock?

Khi giải thuật phát hiện deadlock xác định rằng deadlock đang tồn tại trong hệ thống, có nhiều phương pháp có thể giải quyết deadlock.
Một trong những cách là cho hệ thống tự khôi phục từ deadlock. Có 2 cách để thoát khỏi deadlock :

- Huỷ bỏ một trong các process đang tham gia vào deadlock để phá Circular Wait.
- Trưng dụng (Preempt) một vài tài nguyên đang bị nắm giữ bởi process này cho process kia dùng trước.

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





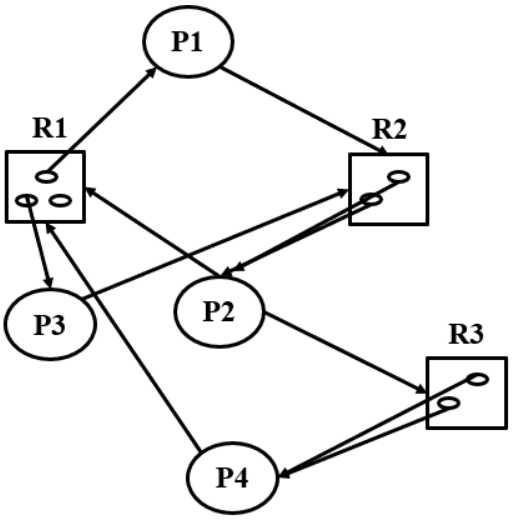
Trả lời:

- 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. (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.
    a. Vẽ đồ thị cấp phát tài nguyên của hệ thống
    b. Hệ thống có deadlock không?
    c. Tìm chuỗi an toàn (nếu có)

Trả lời:

a. Đồ thị cấp phát tài nguyên



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

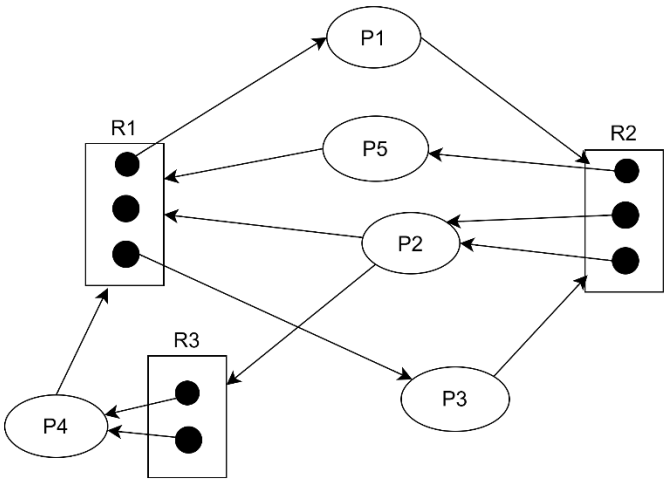
c. Chuỗi an toàn là: <P4, P2, P3, P1> hoặc <P4, P2, P1, P3>.

11. 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.

    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)

Trả lời:

a. Đồ thị cấp phát tài nguyên



b. Hệ thống có 24 chuỗi an toàn.

c. Các chuỗi an toàn:

- <P4, P2, P1, P3, P5>
- <P4, P2, P1, P5, P3>
- <P4, P2, P3, P1, P5>
- <P4, P2, P3, P5, P1>

- <P4, P2, P5, P3, P1>
- <P4, P2, P5, P1, P3>
- <P4, P2, P1, P3, P5>
- <P4, P2, P1, P5, P3>
- <P4, P2, P3, P1, P5>
- <P4, P2, P3, P5, P1>
- <P4, P2, P5, P3, P1>
- <P4, P2, P5, P1, P3>
- <P4, P5, P1, P2, P3>
- <P4, P5, P1, P3, P2>
- <P4, P5, P2, P3, P1>
- <P4, P5, P2, P1, P3>
- <P4, P5, P3, P1, P2>
- <P4, P5, P3, P2, P1>
- <P5, P3, P4, P1, P2>
- <P5, P3, P4, P2, P1>
- <P5, P3, P1, P4, P2>
- <P5, P4, P1, P2, P3>
- <P5, P4, P1, P3, P2>
- <P5, P4, P2, P3, P1>
- <P5, P4, P2, P1, P3>
- <P5, P4, P3, P1, P2>
- <P5, P4, P3, P2, P1>

12. (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:

	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

Available			
A	B	C	D
1	5	2	0

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			
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

Available			
R1	R2	R3	R4
2	1	2	0

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				
Process	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4	
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	P1
P3	0	0	3	4	6	6	5	6	6	6	2	2	4	4	8	6	P4
P4	2	3	5	4	3	3	5	6	1	0	0	2	4	7	11	8	P5
P5	0	3	3	2	0	6	5	2	0	3	2	0	6	7	11	8	P2

Hệ thống có chuỗi an toàn: <P1, P4, P5, P2, P3>

b.

Request P3 (1,1,0,0) ≤ Need P3 (6,6,2,2).

Request P3 (1,1,0,0) ≤ Available thời điểm t1 (2,1,5,2).

Giả sử hệ thống đáp ứng yêu cầu (1,1,0,0) của P3.

Trạng thái mới của hệ thống:

	Allocation				Max				Need				Available				Finish
Process	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4	
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	1	0	3	2	P1
P3	1	1	3	4	6	6	5	6	5	5	2	2	4	3	8	6	P4
P4	2	3	5	4	3	3	5	6	1	0	0	2	4	6	11	8	P5
P5	0	3	3	2	0	6	5	2	0	3	2	0	6	6	11	8	P2

Hệ thống mới vẫn có chuỗi an toàn <P1, P4, P5, P2, P3> cho nên hệ thống đáp ứng yêu cầu cấp phát cho P3 tại thời điểm t1.

14. 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?

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

Trả lời:

a.

	Allocation				Max				Need				Available				Finish
Tiến trình	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
P2	3	1	2	1	3	3	2	1	0	2	0	0	5	6	3	2	P1
P3	0	5	1	0	4	6	1	2	4	1	0	2	5	11	4	2	P3
P4	4	2	1	2	6	3	2	5	2	1	1	3					

Trạng thái không an toàn do không tìm được giá trị Needi nhỏ hơn Available = (5,11,4,2) trong khi Finish[P4] = Finish[P0] = false.

b.

Trạng thái trên an toàn vì hệ thống có chuỗi an toàn: <P1, P2, P3, P4, P0>

	Allocation				Max				Need				Available				Finish
Tiến trình	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	P1
P2	3	1	2	1	3	3	2	1	0	2	0	0	6	3	3	3	P2
P3	0	5	1	0	4	6	1	2	4	1	0	2	6	8	4	3	P3
P4	4	2	1	2	6	3	2	5	2	1	1	3	10	10	5	5	P4

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

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?

	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

Available			
A	B	C	D
3	3	2	1

Trả lời:

a.

	Allocation				Max				Need				Available				Finish
Tiến trình	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	6	4	4	2	P1
P2	2	1	0	3	2	3	1	6	0	2	1	3	8	5	4	5	P2
P3	1	3	1	2	1	4	2	4	0	1	1	2	9	8	5	7	P3
P4	1	4	3	2	3	6	6	5	2	2	3	3	10	12	8	10	P4

Hệ thống an toàn vì có chuỗi an toàn: <P1, P2, P3, P4, P0>

b.

Request P1 (1,1,0,0) ≤ Need P1 (2,1,3,1).

Request P1 (1,1,0,0) ≤ Available thời điểm t0 (3,3,2,1).

Giả sử hệ thống đáp ứng yêu cầu (1,1,0,0) của P1.

Trạng thái mới của hệ thống:

	Allocation				Max				Need				Available				Finish
Tiến trình	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	6	4	4	2	P1
P2	2	1	0	3	2	3	1	6	0	2	1	3	8	5	4	5	P2
P3	1	3	1	2	1	4	2	4	0	1	1	2	9	8	5	7	P3
P4	1	4	3	2	3	6	6	5	2	2	3	3	10	12	8	10	P4

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

c.

Request P4 (0,0,2,0) ≤ Need P4 (2,2,3,3).

Request P4 (0,0,2,0) ≤ Available thời điểm t0 (3,3,2,1).

Giả sử hệ thống đáp ứng yêu cầu (0,0,2,0) của P4.

Trạng thái mới của hệ thống:

	Allocation				Max				Need				Available				
Tiến trình	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	4	2	2	1	5	2	5	2	1	0	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					

Trạng thái không an toàn do không tìm được giá trị Needi nhỏ hơn Available = (3,3,0,1) trong khi Finish[P0] = Finish[P1] = Finish[P2] = Finish[P3] = Finish[P4] = false.