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.

1. Các điều kiện cần để xảy ra deadlock?

* 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.

1. 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:
    - P = {P1, P2,…,Pn} (All process)
    - R = {R1, R2,…,Rn} (All resource)
  + Tập cạnh E gồm 2 loại:
    - Cạnh yêu cầu: Pi -> Rj
    - Cạnh cấp phát: Rj-> Pi
* Mối liên hệ giữa RAG(Resource Allocation Graph)-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

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

* Ngăn chặn deadlock: Ngăn chặn ít nhất một trong 4 điều kiện cần thiết cho deadlock xảy ra bằng cách giới hạn yêu cầu tài nguyên của các quá trình.
  + Ưu điểm: Ngăn chặn hoàn toàn việc xảy ra deadlock, đảm bảo ổn định của hệ thống.
  + Nhược điểm: Có thể gây ra sự chậm trễ trong việc cấp phát tài nguyên, có thể làm giảm hiệu suất.
* Tránh deadlock: Các quá trình cần cung cấp thông tin về tài nguyên mà chúng cần, để hệ thống có thể cấp phát tài nguyên một cách thích hợp và tránh được trạng thái deadlock.
  + Ưu điểm: Hiệu suất hệ thống được giữ nguyên trong quá trình cấp phát tài nguyên.
  + Nhược điểm: Yêu cầu kiến thức sâu về hệ thống để dự đoán và tránh deadlock, có thể dẫn đến sự lãng phí tài nguyên.
* Cho phép hệ thống vào trạng thái deadlock, nhưng sau đó phát hiện deadlock và phục hồi hệ thống.
  + Ưu điểm: Hệ thống vẫn tiếp tục hoạt động mặc dù có deadlock xảy ra, giảm thiểu ảnh hưởng đến hoạt động thông thường.
  + Nhược điểm: Có thể tốn nhiều tài nguyên và thời gian để phát hiện và phục hồi từ deadlock.
* Bỏ qua mọi vấn đề, xem như deadlock không bao giờ xảy ra trong hệ thống. 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 khởi động lại.
  + Ưu điểm: Không tốn nhiều tài nguyên để quản lý hoặc phục hồi từ deadlock.
  + Nhược điểm: Deadlock không được phát hiện có thể dẫn đến giảm hiệu suất của hệ thống. Việc hệ thống ngừng hoạt động và cần khởi động lại có thể xảy ra, ảnh hưởng đến sự ổn định và hiệu suất của hệ thống trong dài hạn.

1. 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 Cứng:
  + Cấp Ngắt (Interrupts):
    - Ưu điểm:
      * Hiệu suất cao: Cấp ngắt giúp tối ưu hóa hiệu suất bằng cách cho phép hệ thống phản hồi ngay lập tức khi có sự kiện xảy ra, không cần tiến trình phải chờ đợi hoặc kiểm tra liên tục.
      * Tương thích với nhiều loại hệ thống: Có thể sử dụng cấp ngắt trong các hệ thống phức tạp và nhiều loại kiến trúc khác nhau.
    - Nhược điểm:
      * Khó triển khai: Việc lập trình và quản lý cấp ngắt có thể phức tạp và yêu cầu kiến thức sâu về phần cứng.
  + Chỉ Thị Test-and-Set Lock (TSL):
    - Ưu điểm:
      * Đơn giản và hiệu quả: Chỉ thị TSL cung cấp một cơ chế đồng bộ đơn giản và hiệu quả, cho phép kiểm tra và thiết lập một biến trong một lần thực thi.
    - Nhược điểm:
      * Busy waiting: Nếu không được xử lý đúng cách, có thể dẫn đến busy waiting, tiêu tốn tài nguyên CPU khi các tiến trình đợi vào vùng tranh chấp.
* Các Giải Pháp Phần Mềm:
  + Sử Dụng Giải Thuật Kiểm Tra Luân Phiên:
    - Ưu điểm:
      * Đơn giản: Giải pháp này đơn giản và dễ triển khai.
      * Không yêu cầu phần cứng đặc biệt: Không yêu cầu tính năng cụ thể từ phần cứng.
    - Nhược điểm:
      * Busy waiting: Tiêu tốn nhiều thời gian CPU vì các tiến trình phải liên tục kiểm tra trạng thái.
  + Sử Dụng Các Biến Cờ Hiệu:
    - Ưu điểm:
      * Đơn giản: Sử dụng biến cờ hiệu để đánh dấu trạng thái của vùng tranh chấp.
    - Nhược điểm:
      * Busy waiting: Cũng có thể dẫn đến busy waiting và tiêu tốn nhiều tài nguyên CPU.
  + Giải Pháp Của Peterson:
    - Ưu điểm:
      * Hiệu quả với số lượng nhỏ tiến trình: Hiệu quả khi chỉ có hai tiến trình tham gia vùng tranh chấp.
    - Nhược điểm:
      * Busy waiting: Có thể gây ra busy waiting khi số lượng tiến trình tăng.
  + Giải Pháp Bakery:
    - Ưu điểm:
      * Quản lý tốt thứ tự truy cập: Hiệu quả trong việc quản lý thứ tự truy cập vào vùng tranh chấp.
    - Nhược điểm:
      * Busy waiting: Có thể dẫn đến busy waiting, không phù hợp với số lượng tiến trình lớn.

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

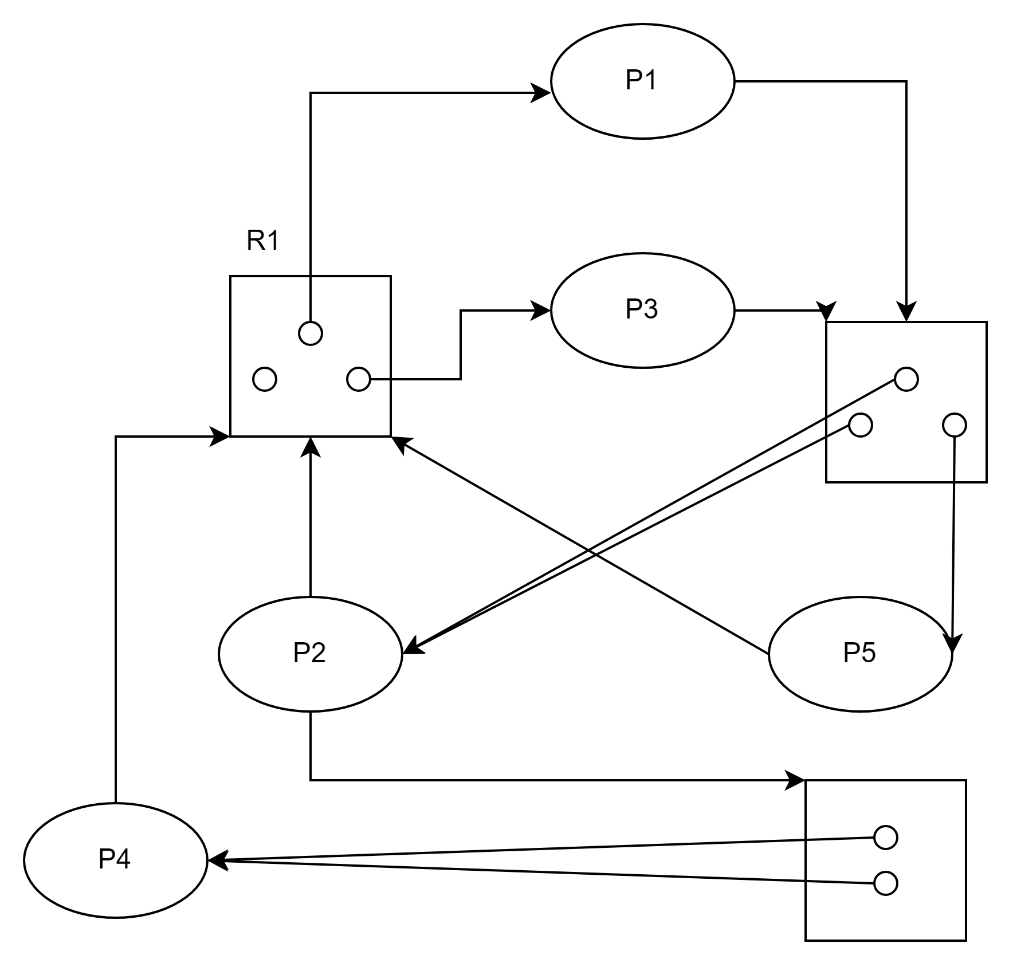
* Một trạng thái của hệ thống được gọi là an toàn (safe) nếu tồn tại một chuỗi thứ tự an toàn. Ví dụ: Một chuỗi tiến trình <P1, P2,…,Pn> là một chuỗi an toàn nếu với mọi i = 1, …, n yêu cầu tối đa về tài nguyên của Pi có thể được thỏa bởi:
  + Tài nguyên mà hệ thống đang có sẵn sàng
  + Cùng với tài nguyên mà tất cả các Pj (j < i) đang giữ
* Một trạng thái của hệ thống được gọi là không an toàn nếu không tồn tại một chuỗi an toàn.
* 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

1. 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?

* Giải thuật an toàn:
  + 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
  + Tìm i thỏa
    - (a) Finish[i] = false
    - (b) Needi ≤ Work (hàng thứ i của Need)
    - Nếu không tồn tại i như vậy, đến bước 4.
  + Work = Work + Allocationi
    - Finish[i] = true
    - quay về bước 2
  + Nếu Finish[i] = true, i = 1,…, n, thì hệ thống đang ở trạng thái safe
* Giải thuật yêu cầu tài nguyên:
  + Request(i) là một vector request của process P(i).
  + Request(i)[j] = k ó P(i) cần k thực thể (instance) của tài nguyên Rj.
  + 1. Nếu Request(i) <= Need(i) thì đến bước 2. Nếu không, báo lỗi vì tiến trình đã vượt yêu cầu tối đa.
  + 2. Nếu Request(i) <= 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 Banker lên hệ thống mới.
    - Nếu hệ thống giả định trên an toàn thì tiến hành cấp phát thực.
    - Nếu trạng thái là không an toàn thì Pi 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 phát hiện deadlock:
  + Đầu tiên ta có các cấu trúc dữ liệu sau :
    - Available : Một vector có độ dài m chỉ số lượng resources của từng loại.
    - Allocation : Một ma trận n x m xác định số resources từng loại đang được cấp cho từng process (có tổng cộng n process).
    - Request : Một ma trận n x m xác định số lượng resources mà từng process yêu cầu thêm ở từng loại resources.
  + Bước 1 : Gọi Work và Finish là hai vector có kích thước lần lượt là m và n. Khởi tạo :
    - Work = Available.
    - Với mọi i = 1,2,…,n, nếu Allocation(i) khác 0 thì Finish[i] = false (tức là còn đang giữ resource thì chưa hoàn thành), ngược lại không giữ resource thì Finish[i] = true.
  + Bước 2 : Tìm i thoả mãn :
    - Finish[i] = false.
    - Request(i) <= Work.
    - Nếu không tồn tại i như vậy, đến bước 4.
  + Bước 3 : Work = Work + Allocation.
    - Finish[i] = true.
    - Quay lại bước 2.
  + Bước 4 : nếu Finish[i] = false, với một số i = 1,2,…,n bất kỳ (tức là khi đến bước 4 vẫn có tiến trình nào đó chưa hoàn thành), thì hệ thống đang ở trạng thái deadlock. Hơn thế nữa, Finish[i] = false thì process P(i) đang bị deadlocked.
  + Thuật toán trên cần m x n^2 phép tính để xác định xem hệ thống có đang bị deadlock hay không.

1. 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.

1. (Bài tập mẫu)
2. (Bài tập mẫu)
3. 

* Có 24 chuỗi an toàn cho hệ thống trên
* <P4, P2, P3, P5, P1>, <P4, P2, P3, P1, P5>, <P4, P2, P5, P3, P1>, <P4, P2, P5, P1, P3>,

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

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

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

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

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

1. (Bài tập mẫu)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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>
* 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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.
* 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 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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>
* 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 |

* 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.