DEADLOCK

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu mô hình hệ thống về deadlock
- Hiểu các đặc điểm của deadlock
- Hiểu các phương pháp quản lý deadlock
- Hiểu cách ngăn chặn deadlock
- Hiểu cách tránh deadlock
- Hiểu cách phát hiện deadlock
- Hiểu cách phục hồi từ deadlock

II Giới thiệu

Trong môi trường đa chương, nhiều quá trình có thể cạnh tranh một số giới hạn tài nguyên. Một quá trình yêu cầu tài nguyên, nếu tài nguyên không sẵn dùng tại thời điểm đó, quá trình đi vào trạng thái chờ. Quá trình chờ có thể không bao giờ chuyển trạng thái trở lại vì tài nguyên chúng yêu cầu bị giữ bởi những quá trình đang chờ khác. Trường hợp này được gọi là deadlock (khoá chết).

Trong chương này chúng ta sẽ mô tả các phương pháp mà hệ điều hành có thể dùng để ngăn chặn hay giải quyết deadlock. Hầu hết các hệ điều hành không cung cấp phương tiện ngăn chặn deadlock nhưng những đặc điểm này sẽ được thêm vào sau đó. Vấn đề deadlock chỉ có thể trở thành vấn đề phổ biến, xu hướng hiện hành gồm số lượng lớn quá trình, chương trình đa luồng, nhiều tài nguyên trong hệ thống và đặc biệt các tập tin có đời sống dài và những máy phục vụ cơ sở dữ liệu hơn là các hệ thống bó.

III Mô hình hệ thống

Một hệ thống chứa số tài nguyên hữu hạn được phân bổ giữa nhiều quá trình cạnh tranh. Các tài nguyên này được phân chia thành nhiều loại, mỗi loại chứa một số thể hiện xác định. Không gian bộ nhớ, các chu kỳ CPU và các thiết bị nhập/xuất (như máy in, đĩa từ) là những thí dụ về loại tài nguyên. Nếu hệ thống có hai CPUs, thì loại tài nguyên CPU có hai thể hiện. Tương tự, loại tài nguyên máy in có thể có năm thể hiện.

Nếu một quá trình yêu cầu một thể hiện của loại tài nguyên thì việc cấp phát bất cứ thể hiện nào của loại tài nguyên này sẽ thoả mãn yêu cầu. Nếu nó không có thì các thể hiện là không xác định và các lớp loại tài nguyên sẽ không được định nghĩa hợp lý. Thí dụ, một hệ thống có thể có hai máy in. Hai loại máy in này có thể được định nghĩa trong cùng lớp loại tài nguyên nếu không có quá trình nào quan tâm máy nào in ra dữ liệu. Tuy nhiên, nếu một máy in ở tầng 9 và máy in khác ở tầng trệt thì người dùng ở tầng 9 không thể xem hai máy in là tương tự nhau và lớp tài nguyên riêng rẻ cần được định nghĩa cho mỗi máy in.

Một quá trình phải yêu cầu một tài nguyên trước khi sử dụng nó, và phải giải phóng sau khi sử dụng nó. Một quá trình có thể yêu cầu nhiều tài nguyên như nó được

yêu cầu để thực hiện tác vụ được gán của nó. Chú ý, số tài nguyên được yêu cầu không vượt quá số lượng tổng cộng tài nguyên sẵn có trong hệ thống. Nói cách khác, một quá trình không thể yêu cầu ba máy in nếu hệ thống chỉ có hai.

Dưới chế độ điều hành thông thường, một quá trình có thể sử dụng một tài nguyên chỉ trong thứ tự sau:

1) **Yêu cầu**: nếu yêu cầu không thể được gán tức thì (thí dụ, tài nguyên đang được dùng bởi quá trình khác) thì quá trình đang yêu cầu phải chờ cho tới khi nó có thể nhận được tài nguyên.
2) **Sử dụng**: quá trình có thể điều hành tài nguyên (thí dụ, nếu tài nguyên là máy in, quá trình có thể in máy in)
3) **Giải phóng**: quá trình giải phóng tài nguyên.

Yêu cầu và giải phóng tài nguyên là các lời gọi hệ thống. Thí dụ như yêu cầu và giải phóng thiết bị, mở và đóng tập tin, cấp phát và giải phóng bộ nhớ. Yêu cầu và giải phóng các tài nguyên khác có thể đạt được thông qua thao tác chờ **wait** và báo hiệu **signal**. Do đó, cho mỗi trường hợp sử dụng, hệ điều hành kiểm tra để đảm bảo rằng quá trình sử dụng yêu cầu và được cấp phát tài nguyên. Một bảng hệ thống ghi nhận mỗi quá trình giải phóng hay được cấp phát tài nguyên. Nếu một quá trình yêu cầu tài nguyên mà tài nguyên đó hiện được cấp phát cho một quá trình khác, nó có thể được thêm vào hàng đợi để chờ tài nguyên này.

Một tập hợp quá trình trong trạng thái deadlock khi mỗi quá trình trong tập hợp này chờ sự kiện mà có thể được tạo ra chỉ bởi quá trình khác trong tập hợp. Những sự kiện mà chúng ta quan tâm chủ yếu ở đây là nhận và giải phóng tài nguyên. Các tài nguyên có thể là tài nguyên vật lý (thí dụ, máy in, đĩa từ, không gian bộ nhớ và chu kỳ CPU) hay tài nguyên luận lý (thí dụ, tập tin, semaphores, monitors). Tuy nhiên, các loại khác của sự kiện có thể dẫn đến deadlock.

Để minh hoạ trạng thái deadlock, chúng ta xét hệ thống với ba ổ đĩa từ. Giả sử mỗi quá trình giữ các một ổ đĩa từ này. Bây giờ, nếu mỗi quá trình yêu cầu một ổ đĩa từ khác thì ba quá trình sẽ ở trong trạng thái deadlock. Mỗi quá trình đang chờ một sự kiện "ổ đĩa từ được giải phóng" mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Thí dụ này minh hoạ deadlock liên quan đến cùng loại tài nguyên.

Deadlock cũng liên quan nhiều loại tài nguyên khác nhau. Thí dụ, xét một hệ thống với một máy in và một ổ đĩa từ. Giả sử, quá trình P_i đang giữ ổ đĩa từ và quá trình P_j đang giữ máy in. Nếu P_i yêu cầu máy in và P_j yêu cầu ổ đĩa từ thì deadlock xảy ra.

Một người lập trình đang phát triển những ứng dụng đa luồng phải quan tâm đặc biệt tới vấn đề này: Các chương trình đa luồng là ứng cử viên cho vấn đề deadlock vì nhiều luồng có thể cạnh tranh trên tài nguyên được chia sẻ.

IV Đặc điểm deadlock

Trong một deadlock, các quá trình không bao giờ hoàn thành việc thực thi và các tài nguyên hệ thống bị buộc chặt, ngăn chặn các quá trình khác bắt đầu. Trước khi chúng ta thảo luận các phương pháp khác nhau giải quyết vấn đề deadlock, chúng ta sẽ mô tả các đặc điểm mà deadlock mô tả.

IV.1 Những điều kiện cần thiết gây ra deadlock

Trường hợp deadlock có thể phát sinh nếu bốn điều kiện sau xảy ra cùng một lúc trong hệ thống:

1) **Loại trừ hỗ tương**: ít nhất một tài nguyên phải được giữ trong chế độ không chia sẻ; nghĩa là, chỉ một quá trình tại cùng một thời điểm có thể sử dụng tài nguyên. Nếu một quá trình khác yêu cầu tài nguyên đó, quá trình yêu cầu phải tạm dừng cho đến khi tài nguyên được giải phóng.
2) **Giữ và chờ cấp thêm tài nguyên**: quá trình phải đang giữ ít nhất một tài nguyên và đang chờ để nhận tài nguyên thêm mà hiện đang được giữ bởi quá trình khác.
3) **Không đòi lại tài nguyên từ quá trình đang giữ chúng**: Các tài nguyên không thể bị đòi lại; nghĩa là, tài nguyên có thể được giải phóng chỉ tự ý bởi quá trình đang giữ nó, sau khi quá trình đó hoàn thành tác vụ.
4) **Tồn tại chu trình trong đồ thị cấp phát tài nguyên**: một tập hợp các quá trình $\{P_0, P_1,\ldots,P_n\}$ đang chờ mà trong đó P_0 đang chờ một tài nguyên được giữ bởi P_1, P_1 đang chờ tài nguyên đang giữ bởi $P_2,\ldots,P_{n-1}$ đang chờ tài nguyên đang được giữ bởi quá trình P_0.

Chúng ta nhấn mạnh rằng tất cả bốn điều kiện phải cùng phát sinh để deadlock xảy ra. Điều kiện chờ đợi ch trình đưa đến điều kiện giữ-và-chờ vì thế bốn điều kiện không hoàn toàn độc lập.

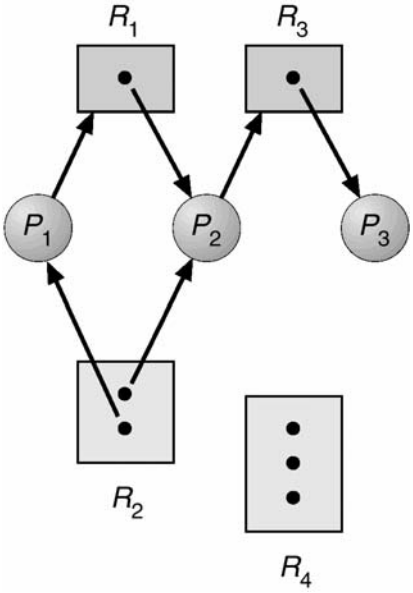
IV.2 Đồ thị cấp phát tài nguyên

Deadlock có thể mô tả chính xác hơn bằng cách hiển thị đồ thị có hướng gọi là đồ thị cấp phát tài nguyên hệ thống. Đồ thị này chứa một tập các đỉnh V và tập hợp các cạnh E. Một tập các đỉnh V được chia làm hai loại nút $P = \{P_1, P_2,\ldots,P_n\}$ là tập hợp các quá trình hoạt động trong hệ thống, và $R = \{R_1, R_2, ..., R_m\}$ là tập hợp chứa tất cả các loại tài nguyên trong hệ thống.

Một cạnh có hướng từ quá trình P_i tới loại tài nguyên R_j được ký hiệu $P_i \rightarrow R_j$; nó biểu thị rằng quá trình P_i đã yêu cầu loại tài nguyên R_j và hiện đang chờ loại tài nguyên đó. Một cạnh có hướng từ loại tài nguyên R_j tới quá trình P_i được hiển thị bởi $R_j \rightarrow P_i$; nó hiển thị rằng thể hiện của loại tài nguyên R_j đã được cấp phát tới quá trình P_i. Một cạnh có hướng $P_i \rightarrow R_j$ được gọi là cạnh yêu cầu; một cạnh có hướng $R_j \rightarrow P_i$ được gọi là cạnh gán.

Bằng hình tượng, chúng ta hiển thị mỗi quá trình P_i là một hình tròn, và mỗi loại tài nguyên R_j là hình chữ nhật. Vì loại tài nguyên R_j có thể có nhiều hơn một thể hiện, chúng ta hiển thị mỗi thể hiện là một chấm nằm trong hình vuông. Chú ý rằng một cạnh yêu cầu trỏ tới chỉ một hình vuông R_j, trái lại một cạnh gán cũng phải gán tới một trong các dấu chấm trong hình vuông.

Khi quá trình P_i yêu cầu một thể hiện của loại tài nguyên R_j, một cạnh yêu cầu được chèn vào đồ thị cấp phát tài nguyên. Khi yêu cầu này có thể được đáp ứng, cạnh yêu cầu lập tức được truyền tới cạnh gán. Khi quá trình không còn cần truy xuất tới tài nguyên, nó giải phóng tài nguyên, và khi đó dẫn đến cạnh gán bị xoá.
Đồ thị cấp phát tài nguyên được hiển thị trong hình VI-1 dưới đây mô tả trường hợp sau:



Hình 0-1 Đồ thị cấp phát tài nguyên

- Các tập P, R, và E:
  - $P = \{P_1, P_2, P_3\}$
  - $R = \{R_1, R_2, R_3, R_4\}$
  - $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$
- Các thể hiện tài nguyên
  - Một thể hiện của tài nguyên loại R_1
  - Hai thể hiện của tài nguyên loại R_2
  - Một thể hiện của tài nguyên loại R_3
  - Một thể hiện của tài nguyên loại R_4
- Trạng thái quá trình
  - Quá trình P_1 đang giữ một thể hiện của loại tài nguyên R_2 và đang chờ một thể hiện của loại tài nguyên R_1.
  - Quá trình P_2 đang giữ một thể hiện của loại tài nguyên R_1 và R_2 và đang chờ một thể hiện của loại tài nguyên R_3.
  - Quá trình P_3 đang giữ một thể hiện của R_3

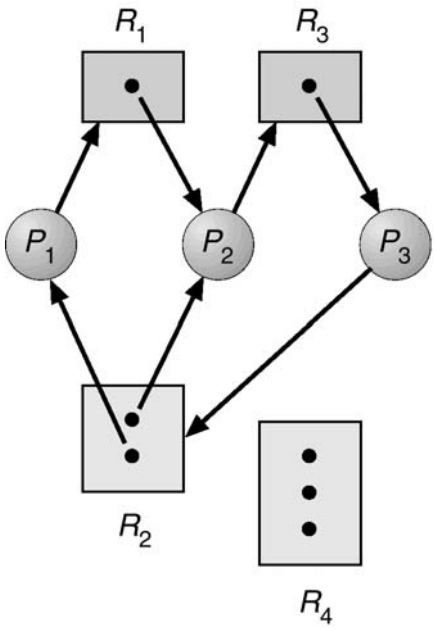
Đồ thị cấp phát tài nguyên hiển thị rằng, nếu đồ thị không chứa chu trình, thì không có quá trình nào trong hệ thống bị deadlock. Nếu đồ thị có chứa chu trình, thì deadlock có thể tồn tại.

Nếu mỗi loại tài nguyên có chính xác một thể hiện, thì một chu trình ngụ ý rằng một deadlock xảy ra. Nếu một chu trình bao gồm chỉ một tập hợp các loại tài nguyên, mỗi loại tài nguyên chỉ có một thể hiện thì deadlock xảy ra. Mỗi quá trình chứa trong chu trình bị deadlock. Trong trường hợp này, 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 thì chu trình không ngụ ý deadlock xảy. Trong trường hợp này, một chu trình trong đồ thị là điều kiện cần nhưng chưa đủ để tồn tại deadlock.

Để hiển thị khái niệm này, chúng ta xem lại đồ thị ở hình VII-1 ở trên. Giả sử quá trình P_3 yêu cầu một thể hiện của loại tài nguyên R_2. Vì không có thể hiện tài nguyên hiện có, một cạnh yêu cầu $P_3 \rightarrow R_2$ được thêm vào đồ thị (hình VI-2). Tại thời điểm này, hai chu trình nhỏ tồn tại trong hệ thống:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

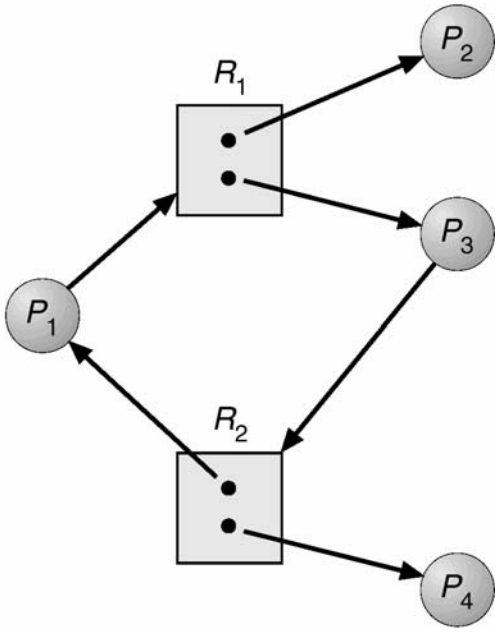


Hình 0-2 Đồ thị cấp phát tài nguyên với deadlock

Quá trình P_1, P_2, và P_3 bị deadlock. Quá trình P_3 đang chờ tài nguyên R_3, hiện được giữ bởi quá trình P_2. Hay nói cách khác, quá trình P_3 đang chờ quá trình P_1 hay P_2 giải phóng tài nguyên R_2. Ngoài ra, quá trình P_1 đang chờ quá trình P_2 giải phóng tài nguyên R_1.

Bây giờ xem xét đồ thị cấp phát tài nguyên trong hình VI-3 dưới đây. Trong thí dụ này, chúng ta cũng có một chu kỳ

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$



Hình 0-3 Đồ thị cấp phát tài nguyên có chu trình nhưng không bị deadlock

Tuy nhiên, không có deadlock. Chú ý rằng quá trình P_4 có thể giải phóng thể hiện của loại tài nguyên R_2. Tài nguyên đó có thể được cấp phát tới P_3 sau đó, chu trình sẽ không còn.

Tóm lại, nếu đồ thị cấp phát tài nguyên không có chu trình thì hệ thống không có trạng thái deadlock. Ngoài ra, nếu có chu trình thì có thể có hoặc không trạng thái deadlock. Nhận xét này là quan trọng khi chúng ta giải quyết vấn đề deadlock.

V Các phương pháp xử lý deadlock

Phần lớn, chúng ta có thể giải quyết vấn đề deadlock theo một trong ba cách:

- Chúng ta có thể sử dụng một giao thức để ngăn chặn hay tránh deadlocks, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock
- Chúng ta có thể cho phép hệ thống đi vào trạng thái deadlock, phát hiện nó và phục hồi.
- Chúng ta có thể bỏ qua hoàn toàn vấn đề này và giả vờ deadlock không bao giờ xảy ra trong hệ thống. Giải pháp này được dùng trong nhiều hệ điều hành, kể cả UNIX.
- Chúng ta sẽ tìm hiểu vắn tắt mỗi phương pháp. Sau đó, chúng ta sẽ trình bày các giải thuật một cách chi tiết trong các phần sau đây.

Để đảm bảo deadlock không bao giờ xảy ra, hệ thống có thể dùng kế hoạch ngăn chặn hay tránh deadlock. Ngăn chặn deadlock là một tập hợp các phương pháp để đảm bảo rằng ít nhất một điều kiện cần (trong phần VI.4.1) không thể xảy ra. Các phương pháp này ngăn chặn deadlocks bằng cách ràng buộc yêu cầu về tài nguyên được thực hiện như thế nào. Chúng ta thảo luận phương pháp này trong phần sau.

Ngược lại, tránh deadlock yêu cầu hệ điều hành cung cấp những thông tin bổ sung tập trung vào loại tài nguyên nào một quá trình sẽ yêu cầu và sử dụng trong thời gian sống của nó. Với những kiến thức bổ sung này, chúng ta có thể quyết định đối với mỗi yêu cầu quá trình nên chờ hay không. Để quyết định yêu cầu hiện tại có thể được thoả mãn hay phải bị trì hoãn, hệ thống phải xem xét tài nguyên hiện có, tài nguyên hiện cấp phát cho mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình.

Nếu một hệ thống không dùng giải thuật ngăn chặn hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống có thể cung cấp một giải thuật để xem xét trạng thái của hệ thống để xác định deadlock có xảy ra hay không và giải thuật phục hồi từ deadlock.

Nếu hệ thống không đảm bảo rằng deadlock sẽ không bao giờ xảy ra và cũng không cung cấp một cơ chế để phát hiện và phục hồi deadlock thì có thể dẫn đến trường hợp hệ thống ở trong trạng thái deadlock. Trong trường hợp này, deadlock không được phát hiện sẽ làm giảm năng lực hệ thống vì tài nguyên đang được giữ bởi những quá trình mà chúng không thể thực thi, đi vào trạng thái deadlock. Cuối cùng, hệ thống sẽ dừng các chức năng và cần được khởi động lại bằng thủ công.

Mặc dù phương pháp này dường như không là tiếp cận khả thi đối với vấn đề deadlock nhưng nó được dùng trong một số hệ điều hành. Trong nhiều hệ thống, deadlock xảy ra không thường xuyên; do đó phương pháp này là rẻ hơn chi phí cho phương pháp ngăn chặn deadlock, tránh deadlock, hay phát hiện và phục hồi deadlock mà chúng phải được sử dụng liên tục. Trong một số trường hợp, hệ thống ở trong trạng thái cô đặc nhưng không ở trạng thái deadlock. Như thí dụ, xem xét một quá trình thời thực chạy tại độ ưu tiên cao nhất (hay bất cứ quá trình đang chạy trên bộ

định thời biểu không trưng dụng) và không bao giờ trả về điều khiển đối với hệ điều hành. Do đó, hệ thống phải có phương pháp phục hồi bằng thủ công cho các điều kiện không deadlock và có thể đơn giản sử dụng các kỹ thuật đó cho việc phục hồi deadlock.

VI Ngăn chặn deadlock

Để deadlock xảy ra, một trong bốn điều kiện cần phải xảy ra. Bằng cách đảm bảo ít nhất một trong bốn điều kiện này không thể xảy ra, chúng ta có thể ngăn chặn việc xảy ra của deadlock. Chúng ta tìm hiểu tỷ mỹ tiếp cận này bằng cách xem xét mỗi điều kiện cần riêng rẻ nhau.

VI.1 Loại trừ hỗ tương

Điều kiện loại trừ hỗ tương phải giữ cho tài nguyên không chia sẻ. Thí dụ, một máy in không thể được chia sẻ cùng lúc bởi nhiều quá trình. Ngược lại, các tài nguyên có thể chia sẻ không đòi hỏi truy xuất loại trừ hỗ tương và do đó không thể liên quan đến deadlock. Những tập tin chỉ đọc là một thí dụ tốt cho tài nguyên có thể chia sẻ. Nếu nhiều quá trình cố gắng mở một tập tin chỉ đọc tại cùng một thời điểm thì chúng có thể được gán truy xuất cùng lúc tập tin. Một quá trình không bao giờ yêu cầu chờ tài nguyên có thể chia sẻ. Tuy nhiên, thường chúng ta không thể ngăn chặn deadlock bằng cách từ chối điều kiện loại trừ hỗ tương: một số tài nguyên về thực chất không thể chia sẻ.

VI.2 Giữ và chờ cấp thêm tài nguyên

Để đảm bảo điều kiện giữ-và-chờ cấp thêm tài nguyên không bao giờ xảy ra trong hệ thống, chúng ta phải đảm bảo rằng bất cứ khi nào một quá trình yêu cầu tài nguyên, nó không giữ bất cứ tài nguyên nào khác. Một giao thức có thể được dùng là đòi hỏi mỗi quá trình yêu cầu và được cấp phát tất cả tài nguyên trước khi nó bắt đầu thực thi. Chúng ta có thể cài đặt sự cung cấp này bằng cách yêu cầu các lời gọi hệ thống yêu cầu tài nguyên cho một quá trình trước tất cả các lời gọi hệ thống khác.

Một giao thức khác cho phép một quá trình yêu cầu tài nguyên chỉ khi quá trình này không có tài nguyên nào. Một quá trình có thể yêu cầu một số tài nguyên và dùng chúng. Tuy nhiên, trước khi nó có thể yêu cầu bất kỳ tài nguyên bổ sung nào, nó phải giải phóng tất cả tài nguyên mà nó hiện đang được cấp phát.

Để hiển thị sự khác nhau giữa hai giao thức, chúng ta xét một quá trình chép dữ liệu từ băng từ tới tập tin đĩa, sắp xếp tập tin đĩa và sau đó in kết quả ra máy in. Nếu tất cả tài nguyên phải được yêu cầu cùng một lúc thì khởi đầu quá trình phải yêu cầu băng từ, tập tin đĩa và máy in. Nó sẽ giữ máy in trong toàn thời gian thực thi của nó mặc dù nó cần máy in chỉ ở giai đoạn cuối.

Phương pháp thứ hai cho phép quá trình yêu cầu ban đầu chỉ băng từ và tập tin đĩa. Nó chép dữ liệu từ băng từ tới đĩa, rồi giải phóng cả hai băng từ và đĩa. Sau đó, quá trình phải yêu cầu lại tập tin đĩa và máy in. Sau đó, chép tập tin đĩa tới máy in, nó giải phóng hai tài nguyên này và kết thúc.

Hai giao thức này có hai nhược điểm chủ yếu. Thứ nhất, việc sử dụng tài nguyên có thể chậm vì nhiều tài nguyên có thể được cấp nhưng không được sử dụng trong thời gian dài. Trong thí dụ được cho, chúng ta có thể giải phóng băng từ và tập tin đĩa, sau đó yêu cầu lại tập tin đĩa và máy in chỉ nếu chúng ta đảm bảo rằng dữ liệu của chúng ta sẽ vẫn còn trên tập tin đĩa. Nếu chúng ta không thể đảm bảo rằng dữ liệu

vẫn còn tập tin đĩa thì chúng ta phải yêu cầu tất cả tài nguyên tại thời điểm bắt đầu cho cả hai giao thức. Thứ hai, đói tài nguyên là có thể. Một quá trình cần nhiều tài nguyên phổ biến có thể phải đợi vô hạn định vì một tài nguyên mà nó cần luôn được cấp phát cho quá trình khác.

VI.3 Không đòi lại tài nguyên từ quá trình đang giữ chúng

Điều kiện cần thứ ba là không đòi lại những tài nguyên đã được cấp phát rồi. Để đảm bảo điều kiện này không xảy ra, chúng ta có thể dùng giao thức sau. Nếu một quá trình đang giữ một số tài nguyên và yêu cầu tài nguyên khác mà không được cấp phát tức thì tới nó (nghĩa là, quá trình phải chờ) thì tất cả tài nguyên hiện đang giữ được đòi lại. Nói cách khác, những tài nguyên này được giải phóng hoàn toàn. Những tài nguyên bị đòi lại được thêm tới danh sách các tài nguyên mà quá trình đang chờ. Quá trình sẽ được khởi động lại chỉ khi nó có thể nhận lại tài nguyên cũ của nó cũng như các tài nguyên mới mà nó đang yêu cầu.

Có một sự chọn lựa khác, nếu một quá trình yêu cầu một số tài nguyên, đầu tiên chúng ta kiểm tra chúng có sẵn không. Nếu tài nguyên có sẵn, chúng ta cấp phát chúng. Nếu tài nguyên không có sẵn, chúng ta kiểm tra chúng có được cấp phát tới một số quá trình khác đang chờ tài nguyên bổ sung. Nếu đúng như thế, chúng ta lấy lại tài nguyên mong muốn đó từ quá trình đang đợi và cấp chúng cho quá trình đang yêu cầu. Nếu tài nguyên không sẵn có hay được giữ bởi một quá trình đang đợi, quá trình đang yêu cầu phải chờ. Trong khi nó đang chờ, một số tài nguyên của nó có thể được đòi lại chỉ nếu quá trình khác yêu cầu chúng. Một quá trình có thể được khởi động lại chỉ khi nó được cấp các tài nguyên mới mà nó đang yêu cầu và phục hồi bất cứ tài nguyên nào đã bị lấy lại trong khi nó đang chờ.

Giao thức này thường được áp dụng tới tài nguyên mà trạng thái của nó có thể được lưu lại dễ dàng và phục hồi lại sau đó, như các thanh ghi CPU và không gian bộ nhớ. Nó thường không thể được áp dụng cho các tài nguyên như máy in và băng từ.

VI.4 Tồn tại chu trình trong đồ thị cấp phát tài nguyên

Điều kiện thứ tư và cũng là điều kiện cuối cùng cho deadlock là điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên. Một cách để đảm bảo rằng điều kiện này không bao giờ xảy ra là áp đặt toàn bộ thứ tự của tất cả loại tài nguyên và đòi hỏi mỗi quá trình trong thứ tự tăng của số lượng.

Gọi $R = \{R_1, R_2, \ldots, R_m\}$ là tập hợp loại tài nguyên. Chúng ta gán mỗi loại tài nguyên một số nguyên duy nhất, cho phép chúng ta so sánh hai tài nguyên và xác định tài nguyên này có đứng trước tài nguyên khác hay không trong thứ tự của chúng ta. Thông thường, chúng ta định nghĩa hàm ánh xạ một-một $F: R \rightarrow N$, ở đây N là tập hợp các số tự nhiên. Thí dụ, nếu tập hợp các loại tài nguyên R gồm các ổ băng từ, ổ đĩa và máy in thì hàm F có thể được định nghĩa như sau:

F(ổ băng từ) = 1,
F(đĩa từ) = 5,
F(máy in) = 12.

Bây giờ chúng ta xem giao thức sau để ngăn chặn deadlock: mỗi quá trình có thể yêu cầu tài nguyên chỉ trong thứ tự tăng của số lượng. Nghĩa là, một quá trình ban đầu có thể yêu cầu bất cứ số lượng thể hiện của một loại tài nguyên R_i. Sau đó, một quá trình có thể yêu cầu các thể hiện của loại tài nguyên R_j nếu và chỉ nếu $F(R_j) > F(R_i)$. Nếu một số thể hiện của cùng loại tài nguyên được yêu cầu, thì một yêu cầu cho tất cả thể hiện phải được cấp phát. Thí dụ, sử dụng hàm được định nghĩa trước đó,

một quá trình muốn dùng ổ băng từ và máy in tại cùng một lúc trước tiên phải yêu cầu ổ băng từ và sau đó yêu cầu máy in.

Nói một cách khác, chúng ta yêu cầu rằng, bất cứ khi nào một quá trình yêu cầu một thể hiện của loại tài nguyên R_j, nó giải phóng bất cứ tài nguyên R_i sao cho $F(R_i) \geq F(R_j)$.

Nếu có hai giao thức được dùng thì điều kiện tồn tại chu trình không thể xảy ra. Chúng ta có thể giải thích điều này bằng cách cho rằng tồn tại chu trình trong đồ thị cấp phát tài nguyên tồn tại. Gọi tập hợp các quá trình chứa tồn tại chu trình trong đồ thị cấp phát tài nguyên là $\{P_0, P_1, \ldots, P_n\}$, ở đây P_i đang chờ một tài nguyên R_i, mà R_i được giữ bởi quá trình P_{i+1}. Vì sau đó quá trình P_{i+1} đang giữ tài nguyên R_i trong khi yêu cầu tài nguyên R_{i+1}, nên chúng ta có $F(R_i) < F(R_{i+1})$ cho tất cả i. Nhưng điều kiện này có nghĩa là $F(R_0) < F(R_1) < \ldots < F(R_n) < F(R_0)$. Bằng qui tắc bắt cầu $F(R_0) < F(R_0)$, điều này là không thể. Do đó, không thể có chờ chu trình.

Chú ý rằng hàm F nên được định nghĩa dựa theo thứ tự tự nhiên của việc sử dụng tài nguyên trong hệ thống. Thí dụ, vì ổ băng từ thường được yêu cầu trước máy in nên có thể hợp lý để định nghĩa F(ổ băng từ) < F(máy in).

VII Tránh deadlock

Các giải thuật ngăn chặn deadlock, được thảo luận ở VII-6, ngăn chặn deadlock bằng cách hạn chế cách các yêu cầu có thể được thực hiện. Các ngăn chặn đảm bảo rằng ít nhất một trong những điều kiện cần cho deadlock không thể xảy ra. Do đó, deadlock không thể xảy ra. Tuy nhiên, các tác dụng phụ có thể ngăn chặn deadlock bởi phương pháp này là việc sử dụng thiết bị chậm và thông lượng hệ thống bị giảm.

Một phương pháp khác để tránh deadlock là yêu cầu thông tin bổ sung về cách tài nguyên được yêu cầu. Thí dụ, trong một hệ thống với một ổ băng từ và một máy in, chúng ta có thể bảo rằng quá trình P sẽ yêu cầu ổ băng từ trước và sau đó máy in trước khi giải phóng cả hai tài nguyên. Trái lại, quá trình Q sẽ yêu cầu máy in trước và sau đó ổ băng từ. Với kiến thức về thứ tự hoàn thành của yêu cầu và giải phóng cho mỗi quá trình, chúng ta có thể quyết định cho mỗi yêu cầu của quá trình sẽ chờ hay không. Mỗi yêu cầu đòi hỏi hệ thống xem tài nguyên hiện có, tài nguyên hiện được cấp tới mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình, để yêu cầu của quá trình hiện tại có thể được thoả mãn hay phải chờ để tránh khả năng xảy ra deadlock.

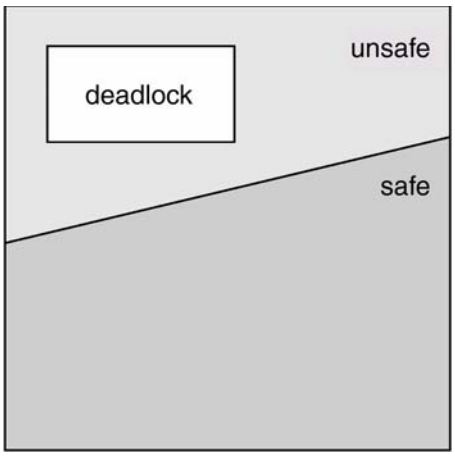
Các giải thuật khác nhau có sự khác nhau về lượng và loại thông tin được yêu cầu. Mô hình đơn giản và hữu ích nhất yêu cầu mỗi quá trình khai báo số lớn nhất tài nguyên của mỗi loại mà nó cần. Thông tin trước về số lượng tối đa tài nguyên của mỗi loại được yêu cầu cho mỗi quá trình, có thể xây dựng một giải thuật đảm bảo hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Đây là giải thuật định nghĩa tiếp cận tránh deadlock. Giải thuật tránh deadlock tự xem xét trạng thái cấp phát tài nguyên để đảm bảo điều kiện tồn tại chu trình trong đồ thị cấp phát tài nguyên có thể không bao giờ xảy ra. Trạng thái cấp phát tài nguyên được định nghĩa bởi số tài nguyên sẵn dùng và tài nguyên được cấp phát và số yêu cầu tối đa của các quá trình.

VII.1 Trạng thái an toàn

Một trạng thái là an toàn nếu hệ thống có thể cấp phát các tài nguyên tới mỗi quá trình trong một vài thứ tự và vẫn tránh deadlock. Hay nói cách khác, một hệ thống ở trong trạng thái an toàn chỉ nếu ở đó tồn tại một thứ tự an toàn. Thứ tự của các quá trình $<P_1, P_2, \ldots, P_n>$ là một thứ tự an toàn cho trạng thái cấp phát hiện hành nếu đối

với mỗi thứ tự P_i, các tài nguyên mà P_i yêu cầu vẫn có thể được thoả mãn bởi tài nguyên hiện có cộng với các tài nguyên được giữ bởi tất cả P_j, với j<i. Trong trường hợp này, nếu những tài nguyên mà quá trình P_i yêu cầu không sẵn dùng tức thì thì P_i có thể chờ cho đến khi tất cả P_j hoàn thành. Khi chúng hoàn thành, P_i có thể đạt được tất cả những tài nguyên nó cần, hoàn thành các tác vụ được gán, trả về những tài nguyên được cấp phát cho nó và kết thúc. Khi P_i kết thúc, P_{i+1} có thể đạt được các tài nguyên nó cần,... Nếu không có thứ tự như thế tồn tại thì trạng thái hệ thống là không an toàn.

Một trạng thái an toàn không là trạng thái deadlock. Do đó, trạng thái deadlock là trạng thái không an toàn. Tuy nhiên, không phải tất cả trạng thái không an toàn là deadlock (hình VI-4). Một trạng thái không an toàn có thể dẫn đến deadlock. Với điều kiện trạng thái là an toàn, hệ điều hành có thể tránh trạng thái không an toàn (và deadlock). Trong một trạng thái không an toàn, hệ điều hành có thể ngăn chặn các quá trình từ những tài nguyên đang yêu cầu mà deadlock xảy ra: hành vi của các quá trình này điều khiển các trạng thái không an toàn.



Hình 0-4 Không gian trạng thái an toàn, không an toàn, deadlock

Để minh hoạ, chúng ta xét một hệ thống với 12 ổ băng từ và 3 quá trình: P_0, P_1, P_2. Quá trình P_0 yêu cầu 10 ổ băng từ, quá trình P_1 có thể cần 4 và quá trình P_2 có thể cần tới 9 ổ băng từ. Giả sử rằng tại thời điểm t_0, quá trình P_0 giữ 5 ổ băng từ, quá trình P_1 giữ 2 và quá trình P_2 giữ 2 ổ băng từ. (Do đó, có 3 ổ băng từ còn rảnh).

	Nhu cầu tối đa	Nhu cầu hiện tại
P0	10	5
P1	4	2
P2	9	2

Tại thời điểm t_0, hệ thống ở trạng thái an toàn. Thứ tự $<P_1,P_0, P_2>$ thoả điều kiện an toàn vì quá trình P_1 có thể được cấp phát tức thì tất cả các ổ đĩa từ và sau đó trả lại chúng (sau đó hệ thống có 5 ổ băng từ sẵn dùng), sau đó quá trình P_0 có thể nhận tất cả ổ băng từ và trả lại chúng (sau đó hệ thống sẽ có 10 ổ băng từ sẵn dùng), và cuối cùng quá trình P_2 có thể nhận tất cả ổ băng từ của nó và trả lại chúng (sau đó hệ thống sẽ có tất cả 12 ổ băng từ sẵn dùng).

Một hệ thống có thể đi từ trạng thái an toàn tới một trạng thái không an toàn. Giả sử rằng tại thời điểm t_1, quá trình P_2 yêu cầu và được cấp 1 ổ băng từ nữa. Hệ thống không còn trong trạng thái an toàn. Tại điểm này, chỉ quá trình P_1 có thể được cấp tất cả ổ băng từ của nó. Khi nó trả lại chúng, chỉ quá trình P_1 có thể được cấp phát tất cả ổ băng từ. Khi nó trả lại chúng, hệ thống chỉ còn 4 ổ băng từ sẵn có. Vì quá

trình P_0 được cấp phát 5 ổ băng từ, nhưng có tối đa 10, quá trình P_0 phải chờ. Tương tự, quá trình P_2 có thể yêu cầu thêm 6 ổ băng từ và phải chờ dẫn đến deadlock.

Lỗi của chúng ta là gán yêu cầu từ quá trình P_2 cho 1 ổ băng từ nữa. Nếu chúng ta làm cho P_2 phải chờ cho đến khi các quá trình khác kết thúc và giải phóng tài nguyên của nó thì chúng ta có thể tránh deadlock.

Với khái niệm trạng thái an toàn được cho, chúng ta có thể định nghĩa các giải thuật tránh deadlock. Ý tưởng đơn giản là đảm bảo hệ thống sẽ luôn còn trong trạng thái an toàn. Khởi đầu, hệ thống ở trong trạng thái an toàn. Bất cứ khi nào một quá trình yêu cầu một tài nguyên hiện có, hệ thống phải quyết định tài nguyên có thể được cấp phát tức thì hoặc quá trình phải chờ. Yêu cầu được gán chỉ nếu việc cấp phát để hệ thống trong trạng thái an toàn.

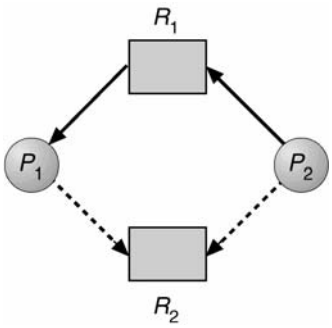
Trong mô hình này, nếu quá trình yêu cầu tài nguyên đang có, nó có thể vẫn phải chờ. Do đó, việc sử dụng tài nguyên có thể chậm hơn mà không có giải thuật tránh deadlock.

VII.2 Giải thuật đồ thị cấp phát tài nguyên

Nếu chúng ta có một hệ thống cấp phát tài nguyên với một thể hiện của mỗi loại, một biến dạng của đồ thị cấp phát tài nguyên được định nghĩa trong phần VI.4.2 có thể được dùng để tránh deadlock.

Ngoài các cạnh yêu cầu và gán, chúng ta giới thiệu một loại cạnh mới được gọi là ***cạnh thỉnh cầu*** (claim edge). Một cạnh thỉnh cầu $P_i \rightarrow R_j$ hiển thị quá trình P_i có thể yêu cầu tài nguyên R_j vào một thời điểm trong tương lai. Cạnh này tương tự cạnh yêu cầu về phương hướng nhưng được hiện diện bởi dấu đứt khoảng. Khi quá trình P_i yêu cầu tài nguyên R_j, cạnh thỉnh cầu $P_i \rightarrow R_j$ chuyển tới cạnh yêu cầu. Tương tự, khi một tài nguyên R_j được giải phóng bởi P_i, cạnh gán $R_j \rightarrow P_i$ được chuyển trở lại thành cạnh thỉnh cầu $P_i \rightarrow R_j$. Chúng ta chú ý rằng các tài nguyên phải được yêu cầu trước trong hệ thống. Nghĩa là, trước khi P_i bắt đầu thực thi, tất cả các cạnh thỉnh cầu của nó phải xuất hiện trong đồ thị cấp phát tài nguyên. Chúng ta có thể giảm nhẹ điều kiện này bằng cách cho phép một cạnh $P_i \rightarrow R_j$ để được thêm tới đồ thị chỉ nếu tất cả các cạnh gắn liền với quá trình P_i là các cạnh thỉnh cầu.

Giả sử rằng P_i yêu cầu tài nguyên R_j. Yêu cầu có thể được gán chỉ nếu chuyển cạnh yêu cầu $P_i \rightarrow R_j$ tới cạnh gán $R_j \rightarrow P_i$ không dẫn đến việc hình thành chu trình trong đồ thị cấp phát tài nguyên. Chú ý rằng chúng ta kiểm tra tính an toàn bằng cách dùng giải thuật phát hiện chu trình. Một giải thuật để phát hiện một chu trình trong đồ thị này yêu cầu một thứ tự của n^2 thao tác, ở đây n là số quá trình trong hệ thống.

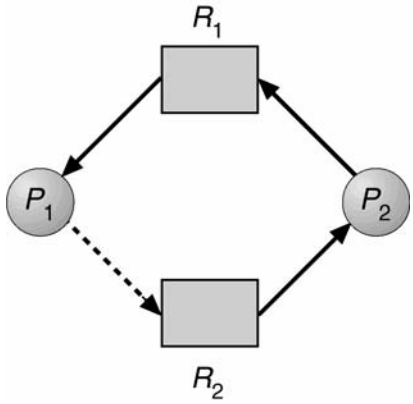


Hình 0-5 Đồ thị cấp phát tài nguyên để tránh deadlock

Nếu không có chu trình tồn tại, thì việc cấp phát tài nguyên sẽ để lại hệ thống trong trạng thái an toàn. Nếu chu trình được tìm thấy thì việc cấp phát sẽ đặt hệ thống

trong trạng thái không an toàn. Do đó, quá trình P_i sẽ phải chờ yêu cầu của nó được thoả.

Để minh hoạ giải thuật này, chúng ta xét đồ thị cấp phát tài nguyên của hình VI-5. Giả sử rằng P_2 yêu cầu R_2. Mặc dù R_2 hiện rảnh nhưng chúng ta không thể cấp phát nó tới P_2 vì hoạt động này sẽ tạo ra chu trình trong đồ thị (Hình VI-6). Một chu trình hiển thị rằng hệ thống ở trong trạng thái không an toàn. Nếu P_1 yêu cầu R_2 và P_2 yêu cầu R_1 thì deadlock sẽ xảy ra.



Hình 0-6 Trạng thái không an toàn trong đồ thị cấp phát tài nguyên

VII.3 Giải thuật của Banker

Giải thuật đồ thị cấp phát tài nguyên không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên. Giải thuật tránh deadlock mà chúng ta mô tả tiếp theo có thể áp dụng tới một hệ thống nhưng ít hiệu quả hơn cơ chế đồ thị cấp phát tài nguyên. Giải thuật này thường được gọi là giải thuật của **Banker**. Tên được chọn vì giải thuật này có thể được dùng trong hệ thống ngân hàng để đảm bảo ngân hàng không bao giờ cấp phát tiền mặt đang có của nó khi nó không thể thoả mãn các yêu cầu của tất cả khách hàng.

Khi một quá trình mới đưa vào hệ thống, nó phải khai báo số tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này có thể không vượt quá tổng số tài nguyên trong hệ thống. Khi một người dùng yêu cầu tập hợp các tài nguyên, hệ thống phải xác định việc cấp phát của các tài nguyên này sẽ để lại hệ thống ở trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ là an toàn, tài nguyên sẽ được cấp; ngược lại quá trình phải chờ cho tới khi một vài quá trình giải phóng đủ tài nguyên.

Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker. Những cấu trúc dữ liệu này mã hoá trạng thái của hệ thống cấp phát tài nguyên. Gọi n là số quá trình trong hệ thống và m là số loại tài nguyên trong hệ thống. Chúng ta cần các cấu trúc dữ liệu sau:

- **Available**: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn dùng của mỗi loại. Nếu Available[j]= k, có k thể hiện của loại tài nguyên R_j sẵn dùng.
- **Max**: một ma trận n x m định nghĩa số lượng tối đa yêu cầu của mỗi quá trình. Nếu Max[i , j] = k, thì quá trình P_i có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên R_j.
- **Allocation**: một ma trận n x m định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp tới mỗi quá trình. Nếu Allocation[i, j] = k, thì quá trình P_i hiện được cấp k thể hiện của loại tài nguyên R_j.

- **Need**: một ma trận n x m hiển thị yêu cầu tài nguyên còn lại của mỗi quá trình. Nếu Need[i, j] = k, thì quá trình P_i có thể cần thêm k thể hiện của loại tài nguyên R_j để hoàn thành tác vụ của nó. Chú ý rằng, Need[i, j] = Max[i, j] – Allocation [i, j].

Cấu trúc dữ liệu này biến đổi theo thời gian về kích thước và giá trị

Để đơn giản việc trình bày của giải thuật Banker, chúng ta thiết lập vài ký hiệu. Gọi X và Y là các vector có chiều dài n. Chúng ta nói rằng $X \le Y$ nếu và chỉ nếu $X[i] \le Y[i]$ cho tất cả i = 1, 2, …, n. Thí dụ, nếu X = (1, 7, 3, 2) và Y = (0, 3, 2, 1) thì $Y \le X$, $Y < X$ nếu $Y \le X$ và $Y \ne X$.

Chúng ta có thể xem xét mỗi dòng trong ma trận Allocation và Need như là những vectors và tham chiếu tới chúng như $Allocation_i$ và $Need_i$ tương ứng. Vector $Allocation_i$ xác định tài nguyên hiện được cấp phát tới quá trình P_i; vector $Need_i$ xác định các tài nguyên bổ sung mà quá trình P_i có thể vẫn yêu cầu để hoàn thành tác vụ của nó.

VII.3.1 Giải thuật an toàn

Giải thuật để xác định hệ thống ở trạng thái an toàn hay không có thể được mô tả như sau:

1) Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available và Finish[i]:=false cho i = 1, 2, …,n.
2) Tìm i thỏa:
   a) Finish[i] = false
   b) $Need_i \le Work$.

   Nếu không có i nào thỏa, di chuyển tới bước 4
3) Work:=Work + $Allocation_i$
   Finish[i] := true
   Di chuyển về bước 2.
4) Nếu Finish[i] = true cho tất cả i, thì hệ thống đang ở trạng thái an toàn.

Giải thuật này có thể yêu cầu độ phức tạp mxn^2 thao tác để quyết định trạng thái là an toàn hay không.

VII.3.2 Giải thuật yêu cầu tài nguyên

Cho $Request_i$ là vector yêu cầu cho quá trình P_i. Nếu $Request_i[j] = k$, thì quá trình P_i muốn k thể hiện của loại tài nguyên R_j. Khi một yêu cầu tài nguyên được thực hiện bởi quá trình P_i, thì các hoạt động sau được thực hiện:

1) Nếu $Request_i \le Need_i$, di chuyển tới bước 2. Ngược lại, phát sinh một điều kiện lỗi vì quá trình vượt quá yêu cầu tối đa của nó.
2) Nếu $Request_i \le Available$, di chuyển tới bước 3. Ngược lại, P_i phải chờ vì tài nguyên không sẵn có.
3) Giả sử hệ thống cấp phát các tài nguyên được yêu cầu tới quá trình P_i bằng cách thay đổi trạng thái sau:

   $Available := Available – Request_i$;
   $Allocation_i := Allocation_i + Request_i$;
   $Need_i := Need_i – Request_i$;

Nếu kết quả trạng thái cấp phát tài nguyên là an toàn, thì giao dịch được hoàn thành và quá trình P_i được cấp phát tài nguyên của nó. Tuy nhiên, nếu trạng thái mới là không an toàn, thì P_i phải chờ $Request_i$ và trạng thái cấp phát tài nguyên cũ được phục hồi.

VII.3.3 Thí dụ minh họa

Xét một hệ thống với 5 quá trình từ P_0 tới P_4, và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 10 thể hiện, loại tài nguyên B có 5 thể hiện và loại tài nguyên C có 7 thể hiện. Giả sử rằng tại thời điểm T_0 trạng thái hiện tại của hệ thống như sau:

	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			

Nội dung ma trận ***Need*** được định nghĩa là ***Max-Allocation*** và là

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

Chúng ta khẳng định rằng hệ thống hiện ở trong trạng thái an toàn. Thật vậy, thứ tự $<P_1, P_3, P_4, P_2, P_0>$ thỏa tiêu chuẩn an toàn. Giả sử bây giờ P_1 yêu cầu thêm một thể hiện loại A và hai thể hiện loại C, vì thế $Request_1 = (1, 0, 2)$. Để quyết định yêu cầu này có thể được cấp tức thì hay không, trước tiên chúng ta phải kiểm tra $Request_1 \leq Available$ (nghĩa là, $(1, 0, 2)) \leq (3, 3, 2)$) là đúng hay không. Sau đó, chúng ta giả sử yêu cầu này đạt được và chúng ta đi đến trạng thái mới sau:

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

Chúng ta phải xác định trạng thái mới này là an toàn hay không. Để thực hiện điều này, chúng ta thực thi giải thuật an toàn của chúng ta và tìm thứ tự $<P_1, P_3, P_4, P_0, P_2>$ thỏa yêu cầu an toàn. Do đó, chúng ta có thể cấp lập tức yêu cầu của quá trình P_1.

Tuy nhiên, chúng ta cũng thấy rằng, khi hệ thống ở trong trạng thái này, một yêu cầu (3, 3, 0) bởi P_4 không thể được gán vì các tài nguyên là không sẵn dùng. Một yêu cầu cho (0, 2, 0) bởi P_0 không thể được cấp mặc dù tài nguyên là sẵn dùng vì trạng thái kết quả là không an toàn.

VIII Phát hiện Deadlock

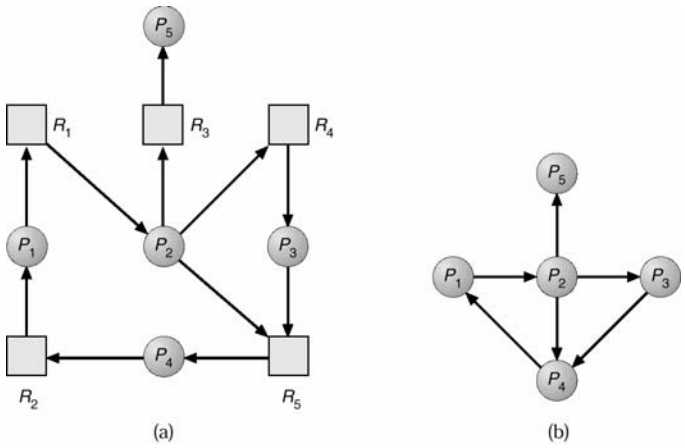
Nếu một hệ thống không thực hiện giải thuật ngăn chặn deadlock hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống phải cung cấp:

- Giải thuật xem xét trạng thái của hệ thống để quyết định deadlock có xảy ra hay không.
- Giải thuật phục hồi từ deadlock

Trong thảo luận dưới đây, chúng ta thảo luận chi tiết về hai yêu cầu khi chúng liên quan đến những hệ thống với chỉ một thể hiện của mỗi loại tài nguyên cũng như đối với hệ thống có nhiều thể hiện cho mỗi loại tài nguyên. Tuy nhiên, tại thời điểm này chúng ta chú ý lược đồ phát hiện và phục hồi yêu cầu chi phí bao gồm không chỉ chi phí tại thời điểm thực thi cho việc duy trì thông tin cần thiết và thực thi giải thuật phát hiện mà còn các lãng phí có thể phát sinh trong việc phát hiện từ deadlock.

VIII.1 Một thể hiện của mỗi loại tài nguyên

Nếu tất cả tài nguyên chỉ có một thể hiện thì chúng ta có thể định nghĩa giải thuật phát hiện deadlock dùng một biến dạng của đồ thị cấp phát tài nguyên, được gọi là đồ thị chờ (wait-for). Chúng ta đạt được đồ thị này từ đồ thị cấp phát tài nguyên bằng cách gỡ bỏ các nút của loại tài nguyên và xóa các cạnh tương ứng.



Hình 0-7 a) Đồ thị cấp phát tài nguyên. b) Đồ thị chờ tương ứng

Chính xác hơn, một cạnh từ P_i tới P_j trong đồ thị chờ hiển thị rằng quá trình P_i đang chờ một quá trình P_j để giải phóng tài nguyên mà P_i cần. Cạnh $P_i \rightarrow P_j$ tồn tại trong đồ thị chờ nếu và chỉ nếu đồ thị cấp phát tài nguyên tương ứng chứa hai cạnh $P_i \rightarrow R_q$ và $R_q \rightarrow P_j$ đối với một số tài nguyên R_q. Thí dụ, trong hình VI-7 dưới đây, chúng ta trình bày đồ thị cấp phát tài nguyên và đồ thị chờ tương ứng.

Như đã đề cập trước đó, deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị chờ chứa chu trình. Để phát hiện deadlock, hệ thống cần duy trì đồ thị chờ và định kỳ gọi giải thuật để tìm kiếm chu trình trong đồ thị.

Một giải thuật phát hiện chu trình trong đồ thị yêu cầu độ phức tạp n^2 thao tác, ở đây n là số cạnh của đồ thị.

VIII.2 Nhiều thể hiện của một loại tài nguyên

Lược đồ đồ thị chờ không thể áp dụng đối với hệ thống cấp phát tài nguyên với nhiều thể hiện cho mỗi loại tài nguyên. Giải thuật phát hiện deadlock mà chúng ta mô tả sau đây có thể áp dụng cho hệ thống này. Giải thuật thực hiện nhiều cấu trúc dữ liệu thay đổi theo thời gian mà chúng tương tự như được dùng trong giải thuật Banker.

- **Available**: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn có của mỗi loại.
- **Allocation**: ma trận nxm định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp phát tới mỗi quá trình.
- **Request**: ma trận nxm hiển thị yêu cầu hiện tại của mỗi quá trình. Nếu Request[i,j] = k, thì quá trình P_i đang yêu cầu k thể hiện nữa của loại tài nguyên R_j.

Quan hệ ≤ giữa hai vectors được định nghĩa trong phần VI.7.3. Để ký hiệu đơn giản, chúng ta sẽ xem những hàng trong ma trận Allocation và Request như các vector, và sẽ tham chiếu chúng như $Allocation_i$ và $Request_i$ tương ứng. Giải thuật phát hiện được mô tả ở đây đơn giản khảo sát mọi thứ tự cấp phát có thể đối với các quá trình còn lại để được hoàn thành. So sánh giải thuật này với giải thuật Banker.

1) Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo Work:=Available. Cho i = 1, 2, …,n, nếu $Allocation_i \neq 0$, thì Finish[i]:= false; ngược lại Finish[i]:= true.
2) Tìm chỉ số i thỏa:
   a) Finish[i] = false
   b) $Request_i \leq Work$.

   Nếu không có i nào thỏa, di chuyển tới bước 4
3) Work:=Work + $Allocation_i$
   Finish[i] := true
   Di chuyển về bước 2.
4) Nếu Finish[i] = false cho một vài i, $1 \leq i \leq n$ thì hệ thống đang ở trạng thái deadlock. Ngoài ra, nếu Finish[i] = false thì quá trình P_i bị deadlock.

Giải thuật này yêu cầu độ phức tạp mxn^2 để phát hiện hệ thống có ở trong trạng thái deadlock hay không.

Câu hỏi đặt ra là tại sao chúng ta trả lại tài nguyên của quá trình P_i (trong bước 3) ngay sau khi chúng ta xác định rằng $Request_i \leq Work$ (trong bước 2b). Chúng ta biết rằng P_i hiện tại không liên quan deadlock (vì $Request_i \leq Work$). Do đó, chúng ta lạc quan và khẳng định rằng P_i sẽ không yêu cầu thêm tài nguyên nữa để hoàn thành công việc của nó; do đó nó sẽ trả về tất cả tài nguyên hiện được cấp phát tới hệ thống. Nếu giả định của chúng ta không đúng, deadlock có thể xảy ra sao đó. Deadlock sẽ được phát hiện tại thời điểm kế tiếp mà giải thuật phát hiện deadlock được nạp.

Để minh hoạ giải thuật này, chúng ta xét hệ thống với 5 quá trình P_0 đến P_4 và 3 loại tài nguyên A, B, C. Loại tài nguyên A có 7 thể hiện, loại tài nguyên B có 2 thể hiện và loại tài nguyên C có 6 thể hiện. Giả sử rằng tại thời điểm T_0, chúng ta có trạng thái cấp phát tài nguyên 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	0			
P3	2	1	1	1	0	0			

P4	0	0	2	0	0	2			

Chúng ta khẳng định rằng hệ thống không ở trong trạng thái deadlock. Thật vậy, nếu chúng ta thực thi giải thuật, chúng ta sẽ tìm ra thứ tự <P_0, P_2, P_3, P_1, P_4> sẽ dẫn đến trong Finish[i] = true cho tất cả i.

Bây giờ giả sử rằng quá trình P_2 thực hiện yêu cầu thêm thể hiện loại C. Ma trận yêu cầu được hiệu chỉnh như sau:

	Need		
	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

Chúng ta khẳng định rằng hệ thống bây giờ bị deadlock. Mặc dù chúng ta có thể đòi lại tài nguyên được giữ bởi quá trình P_0, số lượng tài nguyên sẵn dùng không đủ để hoàn thành các yêu cầu của các quá trình còn lại. Do đó, deadlock tồn tại và bao gồm các quá trình P_1, P_2, P_3, P_4.

VIII.3 Sử dụng giải thuật phát hiện deadlock

Khi nào thì chúng ta nên nạp giải thuật phát hiện deadlock? Câu trả lời phụ thuộc vào hai yếu tố:

1) Deadlock có khả năng xảy ra thường xuyên như thế nào?
2) Bao nhiêu quá trình sẽ bị ảnh hưởng bởi deadlock khi nó sẽ ra?

Nếu deadlock xảy ra thường xuyên thì giải thuật phát hiện nên được nạp lên thường xuyên. Những tài nguyên được cấp phát để các quá trình bị deadlock sẽ rảnh cho đến khi deadlock có thể bị phá vỡ. Ngoài ra, số lượng quá trình liên quan trong chu trình deadlock có thể tăng lên.

Deadlock xảy ra chỉ khi một số quá trình thực hiện yêu cầu mà không được cấp tài nguyên tức thì. Yêu cầu này có thể là yêu cầu cuối hoàn thành một chuỗi các quá trình đang yêu cầu. Ngoài ra, chúng ta có thể nạp giải thuật phát hiện mọi khi một yêu cầu cho việc cấp phát không thể được cấp tức thì. Trong trường hợp này, chúng ta không chỉ định nghĩa tập hợp các quá trình bị deadlock, mà còn xác định quá trình đã gây ra deadlock. (Trong thực tế, mỗi quá trình trong suốt quá trình bị deadlock là một liên kết trong chu trình của đồ thị tài nguyên, vì thế tất cả chúng gây ra deadlock). Nếu có nhiều loại tài nguyên khác nhau, một yêu cầu có thể gây chu trình trong đồ thị tài nguyên, mỗi chu trình hoàn thành bởi yêu cầu mới nhất và "được gây ra" bởi một quá trình có thể xác định.

Dĩ nhiên, nạp giải thuật phát hiện deadlock cho mỗi yêu cầu có thể gây ra một chi phí có thể xem xét trong thời gian tính toán. Một thay đổi ít đắt hơn là nạp giải thuật tại thời điểm ít thường xuyên hơn- thí dụ, một lần một giờ hay bất cứ khi nào việc sử dụng CPU rơi xuống thấp hơn 40%. Nếu giải thuật phát hiện deadlock được nạp trong những thời điểm bất kỳ, thì có nhiều chu trình trong đồ thị tài nguyên. Chúng ta không thể nói quá trình nào của nhiều quá trình bị deadlock gây ra deadlock.

IX Phục hồi deadlock

Khi giải thuật phát hiện xác định rằng deadlock tồn tại, nhiều thay đổi tồn tại. Một khả năng là thông báo người điều hành rằng deadlock xảy ra và để người điều hành giải quyết deadlock bằng thủ công. Một khả năng khác là để hệ thống tự động phục hồi. Có hai tuỳ chọn cho việc phá vỡ deadlock. Một giải pháp đơn giản là huỷ bỏ một hay nhiều quá trình để phá vỡ việc tồn tại chu trình trong đồ thị cấp phát tài nguyên. Tuỳ chọn thứ hai là lấy lại một số tài nguyên từ một hay nhiều quá trình bị deadlock.

IX.1 Kết thúc quá trình

Để xóa deadlock bằng cách hủy bỏ quá trình, chúng ta dùng một trong hai phương pháp. Trong cả hai phương pháp, hệ thống lấy lại tài nguyên được cấp phát đối với quá trình bị kết thúc.

- **Huỷ bỏ tất cả quá trình bị deadlock**: phương pháp này rõ ràng sẽ phá vỡ chu trình deadlock, nhưng chi phí cao; các quá trình này có thể đã tính toán trong thời gian dài, và các kết quả của các tính toán từng phần này phải bị bỏ đi và có thể phải tính lại sau đó.
- **Hủy bỏ một quá trình tại thời điểm cho đến khi chu trình deadlock bị xóa**:phương pháp này chịu chi phí có thể xem xét vì sau khi mỗi quá trình bị hủy bỏ, một giải thuật phát hiện deadlock phải được nạp lên để xác định có quá trình nào vẫn đang bị deadlock.

Hủy bỏ quá trình có thể không dễ. Nếu một quá trình đang ở giữa giai đoạn cập nhật một tập tin, kết thúc nó sẽ để tập tin đó trong trạng thái không phù hợp. Tương tự, nếu quá trình đang ở giữa giai đoạn in dữ liệu ra máy in, hệ thống phải khởi động lại trạng thái đúng trước khi in công việc tiếp theo.

Nếu phương pháp kết thúc một phần được dùng thì với một tập hợp các quá trình deadlock được cho, chúng ta phải xác định quá trình nào (hay các quá trình nào) nên được kết thúc trong sự cố gắng để phá vỡ deadlock. Việc xác định này là một quyết định chính sách tương tự như các vấn đề lập thời biểu CPU. Câu hỏi về tính kinh tế là chúng ta nên hủy bỏ quá trình nào thì sự kết thúc của quá trình đó sẽ chịu chi phí tối thiểu. Tuy nhiên, thuật ngữ chi phí tối thiểu là không chính xác. Nhiều yếu tố có thể xác định quá trình nào được chọn bao gồm:

1) Độ ưu tiên của quá trình là gì.
2) Quá trình đã được tính toán bao lâu và bao lâu nữa quá trình sẽ tính toán trước khi hoàn thành tác vụ được chỉ định của nó.
3) Bao nhiêu và loại tài nguyên gì quá trình đang sử dụng.
4) Bao nhiêu tài nguyên nữa quá trình cần để hoàn thành
5) Bao nhiêu quá trình sẽ cần được kết thúc.
6) Quá trình là giao tiếp hay dạng bó

IX.2 Lấy lại tài nguyên

Để xóa deadlock sử dụng việc trả lại tài nguyên, sau khi chúng ta đòi một số tài nguyên từ các quá trình và cho các tài nguyên này tới các quá trình khác cho đến khi chu trình deadlock bị phá vỡ.

Nếu việc đòi lại được yêu cầu để giải quyết deadlock thì ba vấn đề cần được xác định:

- **Chọn nạn nhân**: những tài nguyên nào và những quá trình nào bị đòi lại? Trong khi kết thúc quá trình, chúng ta phải xác định thứ tự đòi lại để tối thiểu chi phí. Các yếu tố chi phí có thể gồm các tham số như số lượng tài nguyên một quá trình deadlock đang giữ, và lượng thời gian một quá trình deadlock dùng trong sự thực thi của nó.
- **Trở lại trạng thái trước deadlock**: Nếu chúng ta đòi lại tài nguyên từ một quá trình, điều gì nên được thực hiện với quá trình đó? Rõ ràng, nó không thể tiếp tục việc thực thi bình thường; nó đang bị mất một số tài nguyên được yêu cầu. Chúng ta phải phục hồi quá trình tới trạng thái an toàn và khởi động lại từ trạng thái gần nhất trước đó.

Thông thường, rất khó để xác định trạng thái gì là an toàn vì thế giải pháp đơn giản nhất là phục hồi toàn bộ: hủy quá trình và sau đó khởi động lại nó. Tuy nhiên, hữu hiệu hơn để phục hồi quá trình chỉ đủ xa cần thiết để phá vỡ deadlock. Ngoài ra, phương pháp này yêu cầu hệ thống giữ nhiều thông tin hơn về trạng thái của tất cả các quá trình đang chạy.

Đói tài nguyên: chúng ta đảm bảo như thế nào việc đói tài nguyên không xảy ra? Nghĩa là, chúng ta có thể đảm bảo rằng tài nguyên sẽ không luôn bị đòi lại từ cùng một quá trình.

Trong một hệ thống việc chọn nạn nhân ở đâu dựa trên cơ sở các yếu tố chi phí, nó có thể xảy ra cùng quá trình luôn được chọn như là nạn nhân. Do đó, quá trình này không bao giờ hoàn thành tác vụ được chỉ định của nó, một trường hợp đói tài nguyên cần được giải quyết trong bất kỳ hệ thống thực tế. Rõ ràng, chúng ta phải đảm bảo một quá trình có thể được chọn như một nạn nhân chỉ một số lần xác định (nhỏ). Giải pháp chung nhất là bao gồm số lượng phục hồi trong yếu tố chi phí.

X Tóm tắt

Trạng thái deadlock xảy ra khi hai hay nhiều quá trình đang chờ không xác định một sự kiện mà có thể được gây ra chỉ bởi một trong những quá trình đang chờ. Về nguyên tắc, có ba phương pháp giải quyết deadlock:

- Sử dụng một số giao thức để ngăn chặn hay tránh deadlock, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock.
- Cho phép hệ thống đi vào trạng thái deadlock, phát hiện và sau đó phục hồi.
- Bỏ qua vấn đề deadlock và giả vờ deadlock chưa bao giờ xảy ra trong hệ thống. Giải pháp này là một giải pháp được dùng bởi hầu hết các hệ điều hành bao gồm UNIX.

Trường hợp deadlock có thể xảy ra nếu và chỉ nếu bốn điều kiện cần xảy ra cùng một lúc trong hệ thống: loại trừ hỗ tương, giữ và chờ cấp thêm tài nguyên, không đòi lại tài nguyên, và tồn tại chu trình trong đồ thị cấp phát tài nguyên. Để ngăn chặn deadlock, chúng ta đảm bảo rằng ít nhất một điều kiện cần không bao giờ xảy ra.

Một phương pháp để tránh deadlock mà ít nghiêm ngặt hơn giải thuật ngăn chặn deadlock là có thông tin trước về mỗi quá trình sẽ đang dùng tài nguyên như thế nào. Thí dụ, giải thuật Banker cần biết số lượng tối đa của mỗi lớp tài nguyên có thể được

yêu cầu bởi mỗi quá trình. Sử dụng thông tin này chúng ta có thể định nghĩa giải thuật tránh deadlock.

Nếu hệ thống không thực hiện một giao thức để đảm bảo rằng deadlock sẽ không bao giờ xảy ra thì lược đồ phát hiện và phục hồi phải được thực hiện. Giải thuật phát hiện deadlock phải được nạp lên để xác định deadlock có thể xảy ra hay không. Nếu deadlock được phát hiện hệ thống phải phục hồi bằng cách kết thúc một số quá trình bị deadlock hay đòi lại tài nguyên từ một số quá trình bị deadlock.

Trong một hệ thống mà nó chọn các nạn nhân để phụv hồi về trạng thái trước đó chủ yếu dựa trên cơ sở yếu tố chi phí, việc đói tài nguyên có thể xảy ra. Kết quả là quá trình được chọn không bao giờ hoàn thành tác vụ được chỉ định của nó.