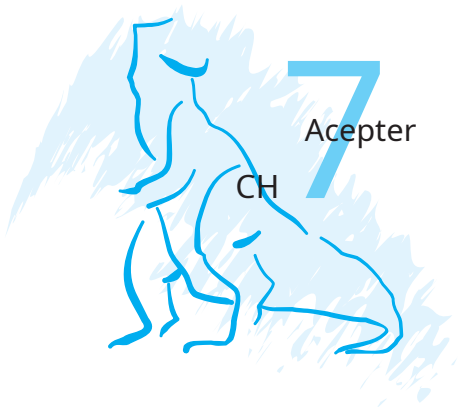

Bế tắc

Bài tập thực hành

7.1 Liệt kê ba ví dụ về các deadlock không liên quan đến môi trường hệ thống máy tính.

Trả lời:

- Hai xe ô tô qua cầu một làn ngược chiều.
- Một người đi xuống thang trong khi một người khác đang leo lên thang.
- Hai đoàn tàu chạy về phía nhau trên cùng một đường ray.
- Hai người thợ mộc phải giã móng tay. Chỉ có một cái búa và một cái gầu đóng đinh. Bế tắc xảy ra nếu một người thợ mộc có cái búa và người thợ mộc kia có cái đinh.

7.2 Giả sử rằng một hệ thống đang ở trạng thái không an toàn. Cho thấy rằng các tiến trình có thể hoàn thành việc thực thi của chúng mà không đi vào trạng thái bế tắc.

Trả lời: Trạng thái không an toàn có thể không nhất thiết dẫn đến deadlock, nó chỉ có nghĩa là chúng ta không thể đảm bảo rằng deadlock sẽ không xảy ra. Do đó, có thể một hệ thống ở trạng thái không an toàn vẫn có thể cho phép tất cả các tiến trình hoàn thành mà không xảy ra bế tắc. Xem xét tình huống trong đó một hệ thống có 12 tài nguyên được phân bổ giữa các quy trình P_0, P_1, và P_2. Các nguồn lực được phân bổ theo chính sách sau:

	Max	Hiện hành	Nhu cầu
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

```
for (int i = 0; i <n; i ++){
    // trước tiên hãy tìm một luồng có thể kết thúc cho
    (int j = 0; j <n; j ++){
        nếu (! kết thúc [j]){
            boolean temp = true;
            for (int k = 0; k <m; k ++){
                if (need [j] [k]> work [k])
                    temp = sai;
            }

            nếu (tạm thời){// nếu chuỗi này có thể kết thúc
                finish [j] = true;
                for (int x = 0; x <m; x ++)
                    work [x] + = work [j] [x];
            }
        }
    }
}
```

Hình 7.1 Thuật toán an toàn của Banker.

Hiện tại có hai tài nguyên có sẵn. Hệ thống này đang ở trạng thái không an toàn vì quá trình P_1 có thể hoàn thành, do đó giải phóng tổng cộng bốn tài nguyên. Nhưng chúng tôi không thể đảm bảo rằng các quy trình P_0 và P_2 có thể hoàn thành. Tuy nhiên, có thể một quy trình có thể giải phóng tài nguyên trước khi yêu cầu thêm. Ví dụ, quy trình P_2 có thể giải phóng một tài nguyên, do đó tăng tổng số tài nguyên lên năm. Điều này cho phép quá trình P_0 để hoàn thành, điều này sẽ giải phóng tổng cộng chín tài nguyên, do đó cho phép quá trình P_2 để hoàn thành.

7.3 Chứng minh rằng thuật toán an toàn được trình bày trong Phần 7.5.3 yêu cầu thứ tự $m \times n^2$ các hoạt động. **Trả lời:**

Hình 7.1 cung cấp mã Java thực thi thuật toán an toàn của thuật toán của ngân hàng (việc triển khai hoàn chỉnh thuật toán của ngân hàng có sẵn với tải xuống mã nguồn).
Có thể thấy, các vòng ngoài lồng nhau — cả hai đều vòng qua n thời gian — cung cấp n^2 màn biểu diễn. Bên trong các vòng bên ngoài này là hai vòng bên trong tuần tự lặp lại m lần. Do đó, điều quan trọng của thuật toán này là $O(m \times n^2)$.

7.4 Hãy xem xét một hệ thống máy tính chạy 5.000 công việc mỗi tháng mà không có kế hoạch ngăn chặn bế tắc hoặc tránh bế tắc. Các lần bế tắc xảy ra khoảng hai lần mỗi tháng và nhà điều hành phải kết thúc và chạy lại khoảng 10 công việc cho mỗi lần bế tắc. Mỗi công việc trị giá khoảng 2 đô la (tính theo thời gian CPU) và các công việc bị chấm dứt có xu hướng hoàn thành một nửa khi chúng bị hủy bỏ.

Một nhà lập trình hệ thống đã ước tính rằng một thuật toán tránh bế tắc (giống như thuật toán của ngân hàng) có thể được cài đặt trong hệ thống với thời gian thực hiện trung bình cho mỗi công việc tăng lên khoảng 10%. Vì máy hiện có 30% thời gian nhàn rỗi, nên tất cả 5.000 công việc mỗi tháng vẫn có thể chạy, mặc dù thời gian quay vòng sẽ tăng trung bình khoảng 20%.

Một. Các đối số để cài đặt thuật toán tránh bế tắc là gì?

b. Các lập luận chống lại việc cài đặt thuật toán tránh bế tắc là gì?

Trả lời: Một lập luận để cài đặt tính năng tránh deadlock trong hệ thống là chúng tôi có thể đảm bảo deadlock sẽ không bao giờ xảy ra. Ngoài ra, mặc dù thời gian quay vòng tăng lên, tất cả 5.000 công việc vẫn có thể chạy.
Một lập luận chống lại việc cài đặt phần mềm tránh deadlock là các deadlock xảy ra không thường xuyên và chúng tốn rất ít chi phí khi chúng xảy ra.

7,5 Hệ thống có thể phát hiện ra rằng một số quy trình của nó đang bị chết đói không? Nếu bạn trả lời "có", hãy giải thích cách nó có thể. Nếu bạn trả lời "không", hãy giải thích cách hệ thống có thể giải quyết vấn đề chết đói.
Trả lời: Đói là một chủ đề khó xác định vì nó có thể có những ý nghĩa khác nhau đối với các hệ thống khác nhau. Đối với mục đích của câu hỏi này, chúng tôi sẽ định nghĩa chết đói là tình huống theo đó một quy trình phải đợi sau một khoảng thời gian hợp lý — có thể là vô thời hạn — trước khi nhận được tài nguyên được yêu cầu. Một cách để phát hiện nạn đói trước tiên là xác định một khoảng thời gian— *T*- điều đó được coi là không hợp lý. Khi một tiến trình yêu cầu một tài nguyên, một bộ đếm thời gian được bắt đầu. Nếu thời gian trôi qua vượt quá *T*, thì quá trình này được coi là bị bỏ đói.
Một chiến lược để đối phó với nạn đói sẽ là áp dụng một chính sách trong đó các nguồn lực chỉ được giao cho quá trình được chờ đợi lâu nhất. Ví dụ, quy trình if $P_{Một}$ đã chờ đợi tài nguyên lâu hơn *X* hơn quá trình P_b, yêu cầu từ quá trình P_b sẽ được hoãn lại cho đến khi xử lý $P_{Một}$' yêu cầu của s đã được đáp ứng.
Một chiến lược khác sẽ ít nghiêm ngặt hơn những gì vừa được đề cập. Trong trường hợp này, một tài nguyên có thể được cấp cho một quá trình chờ đợi ít hơn một quá trình khác, với điều kiện là quá trình kia không bị chết đói. Tuy nhiên, nếu một quá trình khác được coi là đang chết đói, yêu cầu của nó trước tiên sẽ được đáp ứng.

7.6 Hãy xem xét chính sách phân bổ nguồn lực sau đây. Yêu cầu và phát hành tài nguyên được cho phép bất cứ lúc nào. Nếu yêu cầu tài nguyên không thể được đáp ứng vì tài nguyên không có sẵn, thì chúng tôi sẽ kiểm tra bất kỳ quy trình nào bị chặn, đang chờ tài nguyên. Nếu họ có các tài nguyên mong muốn, thì các tài nguyên này sẽ bị lấy đi và được đưa cho quá trình yêu cầu. Vectơ tài nguyên mà quá trình đang chờ được tăng lên để bao gồm các tài nguyên đã bị lấy đi.

Ví dụ: hãy xem xét một hệ thống có ba loại tài nguyên và vectơ *Có sẵn* được khởi tạo thành (4,2,2). Nếu quá trình P_0 yêu cầu (2,2,1), nó nhận được chúng. Nếu P_1 yêu cầu (1,0,1), nó nhận được chúng. Sau đó nếu P_0 yêu cầu (0,0,1), nó bị chặn (tài nguyên không có sẵn). Nếu P_2 bây giờ yêu cầu (2,0,0), nó nhận được một cái có sẵn (1,0,0) và một cái đã được phân bổ cho P_0 (từ P_0 bị chặn). P_0'S *Phân bổ* vectơ đi xuống (1,2,1) và *Nhu cầu* vectơ đi lên (1,0,1).

Một. Có thể xảy ra bế tắc không? Nếu bạn trả lời "có", hãy đưa ra một ví dụ. Nếu bạn trả lời "không", hãy chỉ định điều kiện cần thiết nào không thể xảy ra.

b. Chặn vô thời hạn có thể xảy ra không? Giải thich câu trả lơi của bạn.

Trả lời:

Một. Bế tắc không thể xảy ra vì quyền ưu tiên tồn tại.

b. Vâng. Một quy trình có thể không bao giờ có được tất cả các tài nguyên mà nó cần nếu chúng liên tục bị tấn công bởi một loạt các yêu cầu như quy trình *C*.

7.7 Giả sử rằng bạn đã viết mã thuật toán an toàn tránh bế tắc và bây giờ được yêu cầu triển khai thuật toán phát hiện bế tắc. Bạn có thể làm như vậy chỉ bằng cách sử dụng mã thuật toán an toàn và xác định lại $Ma\ x_{tôi} = Chờ_{tôi} + Alloca\ ti\ trên_{tôi}$, ở đâu $Chờ_{tôi}$ là một vectơ chỉ định quy trình tài nguyên *tôi* đang chờ đợi, và $Alloca\ ti\ trên_{tôi}$ như được định nghĩa trong Phần 7.5? Giải thích câu trả lơi của bạn.

Trả lời:

Vâng. Các *Max* vectơ đại diện cho yêu cầu tối đa mà một quy trình có thể thực hiện. Khi tính toán thuật toán an toàn, chúng tôi sử dụng *Nhu cầu* ma trận, đại diện cho *Max - Phân bổ*. Một cách khác để nghĩ về điều này là *Tối đa = Cần + Phân bổ*. Theo câu hỏi, *Chờ* ma trận thực hiện một vai trò tương tự như *Nhu cầu* ma trận, do đó *Tối đa = Chờ đợi + Phân bổ*.

7.8 Có thể có một bế tắc chỉ liên quan đến một quy trình duy nhất không? Giải thich câu trả lơi của bạn.

Trả lời: Không. Điều này xảy ra trực tiếp từ điều kiện giữ và chờ.