MÔN HỆ ĐIỀU HÀNH

Chương 4

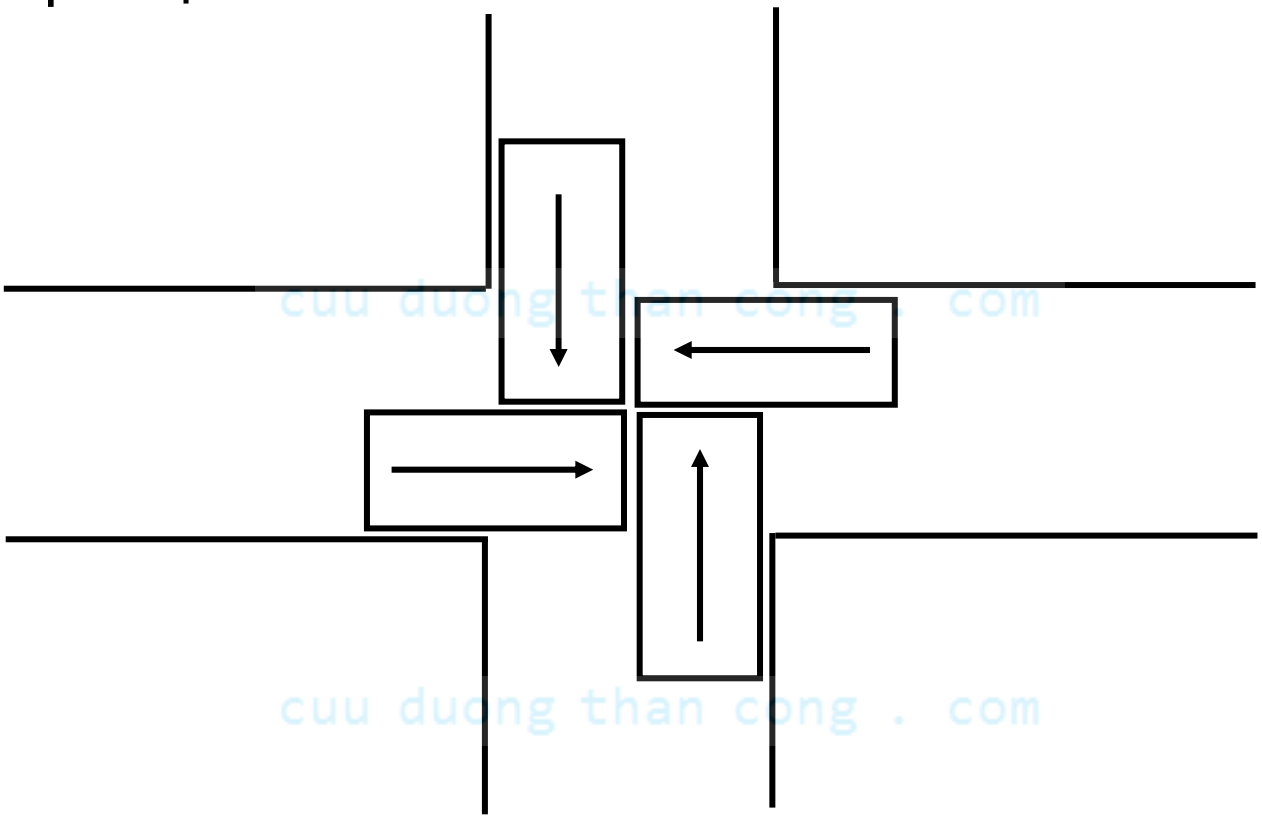
DEADLOCK &XỬ LÝ

4.1 Định nghĩa deadlock
4.2 Bốn điều kiện cần và đủ để gây ra deadlock
4.3 Bốn chiến lược giải quyết deadlock
4.4 Chiến lược phát hiện & chữa trị deadlock
4.5 Chiến lược né tránh deadlock
4.6 Chiến lược phòng ngừa deadlock

Tài liệu tham khảo : chương 2, sách "Modern Operating Systems", Andrew S. Tanenbaum: , 2nd ed, Prentice Hall

4.1 Định nghĩa deadlock

- Deadlock là trạng thái của hệ thống mà ở đó có ít nhất 2 process đang dừng chờ lẫn nhau và như thế chúng không thể chạy tiếp được.



4.2 Bốn điều kiện cần và đủ để gây ra deadlock

1. Loại trừ tương hỗ đoạn code CS truy xuất tài nguyên dùng chung của các process chạy đồng thời.
2. Process giữ tài nguyên cũ đang chiếm dụng trong khi cố gắng xin thêm tài nguyên mới.
3. Hệ thống có dùng tài nguyên "non-preemptive", là loại tài nguyên mà sau khi đã giao cho 1 process nào đó truy xuất, hệ thống không được quyền lấy lại tạm thời để cho process khác truy xuất.
4. Đã xuất hiện vòng khép kín giữa các process chờ nhau.

4.3 Bốn chiến lược giải quyết deadlock

1. Phớt lờ : không làm gì cả vì hy vọng hệ thống sẽ không có deadlock ➔ Nếu hệ thống có deadlock thì chịu chết!!.
2. Phát hiện và chữa trị deadlock (Dectection & Recovery) : cứ để hệ thống hoạt động tự do, theo định kỳ hay khi hệ thống rãnh, máy sẽ kiểm tra để phát hiện có deadlock không ? Nếu không thì thôi, nếu có thì tìm cách chữa trị sao cho hệ thống hết bị deadlock và làm việc bình thường trở lại.
3. Né tránh deadlock (Deadlock Avoidance) : mỗi khi sắp cấp phát tài nguyên cho process, máy kiểm tra cẩn thận xem có dẫn đến deadlock không ? Nếu không thì cấp phát bình thường, còn nếu có nguy cơ deadlock thì trì hoãn việc cấp phát để né tránh deadlock có thể xảy ra.
4. Phòng ngừa deadlock (deadlock prevention) : hệ thống sẽ dùng 1 tập các nguyên tắc rất nghiêm khắc trong việc cấp phát tài nguyên cho các process sao cho deadlock không thể xảy ra.

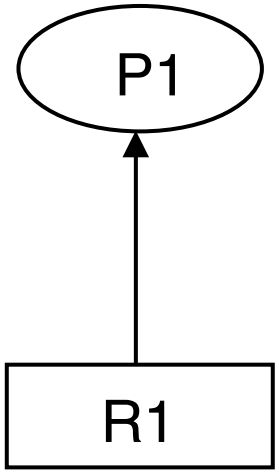
4.4 Chiến lược phát hiện & chữa trị deadlock

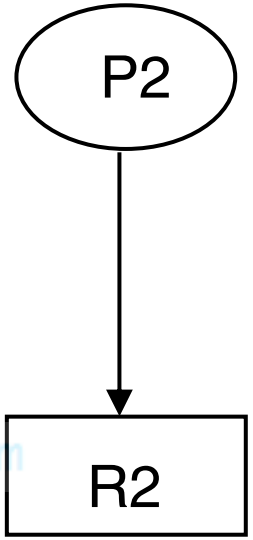
1. Giải thuật đơn giản cho hệ thống mà mỗi loại tài nguyên chỉ có tối đa 1 tài nguyên (hệ thống có 1 CPU, 1 đĩa cứng, 1 máy in, 1 scanner,...).
2. Giải thuật tổng quát cho hệ thống mà mỗi loại tài nguyên có thể có nhiều tài nguyên (hệ thống có 8 CPU, 4 đĩa cứng, 5 máy in, 3 scanner,...)

Giải thuật phát hiện deadlock đơn giản

- Nếu mỗi loại tài nguyên chỉ có tối đa 1 tài nguyên thì khi 1 process Pi nào đó đang chiếm giữ tài nguyên Rj thì không có process nào khác có thể truy xuất Rj nữa (nguyên tắc loại trừ tương hỗ). Như vậy, nếu process Pj cần truy xuất Rj, nó buộc phải dừng chờ process Pi trả tài nguyên Rj, ta nói trong trường hợp này, Pj phụ thuộc Pi.
- Ý tưởng cơ bản của giải thuật phát hiện deadlock đơn giản là hệ thống sẽ xây dựng và quản lý đồ thị miêu tả sự phụ thuộc giữa các process theo thời gian. Đồ thị này có các thành phần sau : mỗi process hay mỗi tài nguyên là 1 nút của đồ thị, cung có hướng từ nút i đến j miêu tả phần tử i phụ thuộc phần tử j.

Giải thuật phát hiện deadlock đơn giản





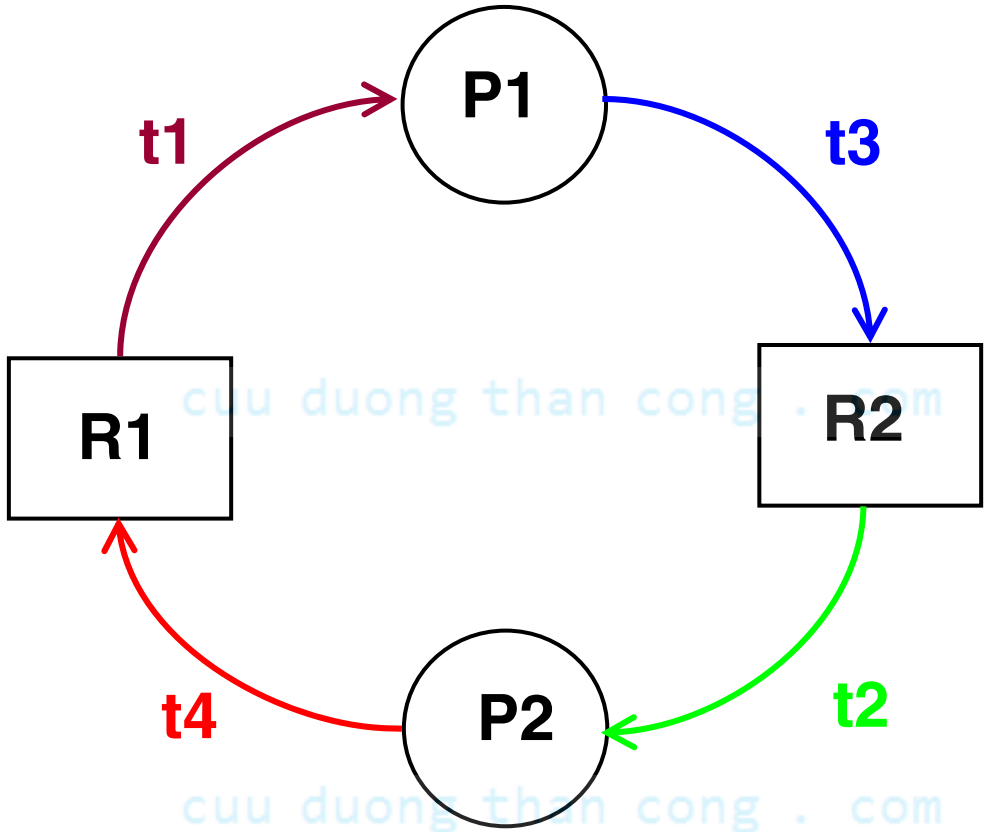
Tài nguyên R1 đang bị P1 truy xuất (chiếm giữ)

Process P2 đang dừng chờ tài nguyên R2 (đang bị chiếm giữ bởi process khác)

Giải thuật phát hiện deadlock đơn giản

- Thí dụ có 2 process P1 và P2 đang chạy, theo giải thuật process P1 sẽ truy xuất tài nguyên R1 rồi R2, trong khi đó process P2 sẽ truy xuất R2 rồi R1 với tiến độ thời gian cụ thể như sau :
  - tại t1 : process P1 xin truy xuất R1 $\Rightarrow$ OK $\Rightarrow$ hệ thống vẽ 1 cung từ R1 tới P1 và cho P1 chạy tiếp.
  - tại t2 : process P2 xin truy xuất R2 $\Rightarrow$ OK $\Rightarrow$ hệ thống vẽ 1 cung từ R2 tới P2 và cho P2 chạy tiếp.
  - tại t3 : process P1 xin truy xuất R2 (đang bị P2 chiếm giữ) $\Rightarrow$ hệ thống vẽ 1 cung từ P1 tới R2 và bắt P1 dừng đợi process P2.
  - tại t4 : process P2 xin truy xuất R1 (đang bị P1 chiếm giữ) $\Rightarrow$ hệ thống vẽ 1 cung từ P2 tới R1 và bắt P2 dừng đợi process P1.
  - từ t4 trở đi : đã xuất hiện vòng kép kín chứa 2 process P1 và P2 $\Rightarrow$ 2 process P1 và P2 đều bị dừng vì phải chờ lẫn nhau và chúng không bao giờ chạy được nữa $\Rightarrow$ deadlock.

Giải thuật phát hiện deadlock đơn giản



4.5 Giải thuật phát hiện deadlock tổng quát

Trạng thái sử dụng các tài nguyên của các process tại từng thời điểm được xác định bởi 4 thông số sau đây :

- vector miêu tả số lượng tài nguyên tổng thể đã được gắn vào máy E(E1, E2,...,Em), trong đó Ei là số lượng tài nguyên loại i mà hệ thống được trang bị. Như vậy vector E là hằng số trong lúc hệ thống hoạt động (ta không được phép gắn/gở tài nguyên trong lúc máy đang vận hành).
- vector miêu tả số lượng tài nguyên chưa dùng (đang rãnh) A(A1, A2,...,Am), trong đó Ai là số lượng tài nguyên loại i còn đang rãnh. Như vậy, vector **A** ≤ **E** theo nghĩa ∀j, Aj ≤ Ej.

4.5 Giải thuật phát hiện deadlock tổng quát

- ma trận C miêu tả số lượng tài nguyên thuộc từng loại đã được cấp phát cho các process (giả sử có n process và m loại tài nguyên khác nhau) :

$$\begin{pmatrix} C11 & C12 & C13 & \dots & C1m \\ C21 & C22 & C23 & \dots & C2m \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Cn1 & Cn2 & Cn3 & \dots & Cnm \end{pmatrix}$$

Phần tử Cij miêu tả số lượng tài nguyên loại j đang bị process i chiếm giữ.

4.5 Giải thuật phát hiện deadlock tổng quát

- ma trận R miêu tả số lượng tài nguyên thuộc từng loại đang được các process xin thêm nhưng chưa được cấp phát (vì chưa có sẵn!) :

$$\begin{pmatrix} R11 & R12 & R13 & \dots & R1m \\ R21 & R22 & R23 & \dots & R2m \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Rn1 & Rn2 & Rn3 & \dots & Rnm \end{pmatrix}$$

Phần tử Rij miêu tả số lượng tài nguyên loại j đang được process i xin thêm nhưng chưa được cấp phát.

4.5 Giải thuật phát hiện deadlock tổng quát

$$E_j = A_j + \sum_{i=1}^{n} C_{ij}$$

Số tài nguyên tổng thể loại j = Số tài nguyên loại j còn rãnh + Tổng Số tài nguyên loại j đang bị n process chiếm giữ

4.5 Giải thuật phát hiện deadlock tổng quát

BOOL deadlock[n]; // = 0 : chua deadlock, = 1 : bi deadlock

int E[m]; // bản sao vector E của hệ thống

Int A[m]; // bản sao vector A của hệ thống

int C[n][m]; // bản sao vector E của hệ thống

int R[n][m]; // bản sao vector E của hệ thống

4.5 Giải thuật phát hiện deadlock tổng quát

```
// dùng timer định kỳ tạo ngắt để chạy thủ tục này
void Deadlock_Detection(void) {
int i;
        //1. khởi động các process đều bị deadlock
        for (i=0; i<n; i++) deadlock[i] = 1;
        //2. Lặp tìm process i chạy được
        while (i=FindProcess()) >=0) { // có process chạy được
                deadlock[i] = 0;
                for (j = 0; j<m;j++)  A[j] += C[i][j];
        }
        //3. kiểm tra có deadlock không
        for (i=0; i<n,i++) if (deadlock[i] ==1) break;
        if (i>=n) return; // không có deadlock
        // có deadlock, giải quyết
        ....
}
```

4.5 Giải thuật phát hiện deadlock tổng quát

```
// Tìm 1 process có thể chạy được
int FindProcess(void) {
        for (i=0; i<n;i++)
                if (deadlock[i]==1 && lessequal(R,i,A)) return i;
        // không tìm được process có thể chạy tiếp
        return -1;
}

// Kiểm tra hàng i của ma trận C có nhỏ hơn hay bằng vector A
BOOL lessequal(C,i,A) {
        for (j=0;j<m;j++) if (C[i][j] > A[j]) return FALSE;
        return TRUE;
}
```

4.5 Giải thuật phát hiện deadlock tổng quát

Chữa trị deadlock :

- dùng cơ chề "Preemption" tài nguyãn có liên quan : không tổng quát vì không khả thi nếu tài nguyêôn liên quan là tài nguyên dạng “non-preemptive”.
- giết 1 hay n process rồi cho chúng chạy lại từ đầu : chọn process nào để giết? Giết process sẽ làm mất tính nhất quán dữ liệu (vì bị process hiệu chỉnh 1 phần rồi).
- rollback process về checkpoint thích hợp : tốn bộ nhớ lưu trữ trạng thái process ở những checkpoint.

4.6 Chiến lược né tránh deadlock

Né tránh deadlock :

- để hệ thống hoạt động tự do, chỉ khi cần cấp phát tài nguyên cho process thì phải cẩn thận : tiên tri xem nếu cấp phát thì có dẫn hệ thống đến deadlock không ? Nếu có thì không cấp phát (cho process xin cấp phát ngủ chờ), nều không thì cấp phát tài nguyên bình thường.
- Trạng thái an toàn là trạng thái mà ở đó chưa có deadlock và tồn tại ít nhất 1khả năng chạy các process sao cho chúng hoàn tất chức năng. Ngược lại ta nói hệ thống đang ở trạng thái không an toàn.
- Như vậy, trạng thái bắt đầu của hệ thống (E^0,A^0,C^0,R^0) là trạng thái an ntoàn.

4.6 Chiến lược né tránh deadlock

- Nếu hệ thống đang ở trạng thái an toàn (E^i,A^i,C^i,R^i), nếu có process giải phóng tài nguyên thì hệ thống sẽ chuyển đến trạng thái $(E^{i+1},A^{i+1},C^{i+1},R^{i+1})$, an toàn hơn $\Rightarrow$ không cần kiểm tra trường hợp này.
- Nếu hệ thống đang ở trạng thái an toàn (E^i,A^i,C^i,R^i), nếu có process xin cấp phát tài nguyên thì hệ thống sẽ chuyển đến trạng thái $(E^{i+1},A^{i+1},C^{i+1},R^{i+1})$, trạng thái này có thể an toàn hoặc không an toàn $\Rightarrow$ cần kiểm tra cẩn thận trường hợp này : dùng thuật giải phát hiện deadlock trên trạng thái $(E^{i+1},A^{i+1},C^{i+1},R^{i+1})$ xem có bị deadlock không? Nếu có thì không cấp phát (cho process xin cấp phát ngủ chờ), nếu không thì cấp phát tài nguyên bình thường.

4.7 Chiến lược đề phòng deadlock

Tìm cách cấp phát tài nguyên sao cho về nguyên tắc không thể gây ra deadlock về sau :

- Ngừa nguyên nhân 1 : đừng tạo cơ hội cho các process phải loại trừ tương hỗ lẫn nhau khi thi hành đoạn code CS truy xuất tài nguyên dùng chung, thí dụ dùng kỹ thuật 'Spooling' máy in. Kỹ thuật này không có tính tổng quát cao vì không thích hợp cho mọi tài nguyên.

4.7 Chiến lược đề phòng deadlock

Ngừa nguyên nhân 2 : đừng để process giữ tài nguyên cũ (đang chiếm giữ) khi xin và chờ tài nguyên mới :

- cho mỗi process dùng đúng 1 tài nguyên, như vậy nếu process đang xin tài nguyên mà chưa cấp phát thì không process nào chờ nó cả, còn nếu process đã được cấp phát tài nguyên, nó không được xin thêm nên không thể chờ process khác được.
- cho mỗi process dùng tự do n tài nguyên, nhưng phải xin toàn bộ 1 lần, không được xin nhiều lần $\Rightarrow$ cũng không gây ra việc các process chờ vòng lẫn nhau.
- cho mỗi process dùng tự do n tài nguyên và được phép xin nhiều lần theo yêu cầu của thuật giải riêng, nhưng mỗi lần xin tài nguyên mới, process phải trả lại tất cả các tài nguyên đang chiếm giữ rồi xin lại cùng với tài nguyên mới $\Rightarrow$ cũng không gây ra việc các process chờ vòng lẫn nhau.

4.7 Chiến lược đề phòng deadlock

- Ngừa nguyên nhân 3 : đừng sử dụng tài nguyên dạng "No preemptive" $\Rightarrow$ không khả thi.
- Ngừa nguyên nhân 4 : đừng để các process tạo vòng chờ khép kín : đánh số thứ tự các tài nguyên rồi yêu cầu các process phải xin cấp phát các tài nguyên theo thứ tự xác định (tăng dần hay giảm dần). Vấn đề là đánh số các tài nguyên theo thứ tự nào cho hợp lý để mọi process đều có thể hoạt động theo thuật giải riêng của mình?