# Tình trạng bế tắc

Sunday, September 13, 2020 3:11 PM

## Khái niệm về bế tắc

### Định nghĩa

Bế tắc (deadlock) là tình trạng ganh đua tài nguyên mà một nhóm tiến trình giữ và chờ tài nguyên lẫn nhau, không có tiến trình nào có đủ tài nguyên cần thiết để hoàn tất công việc.

**Ví dụ**: Bài toán bữa ăn của các nhà hiền triết, mỗi người đều lấy được đũa trái và chờ lấy đũa còn lại. Tất cả đều không ăn được.

### Điều kiện cần và đủ gây ra deadlock

Deadlock xảy ra *khi và chỉ khi* 4 điều kiện sau đồng thỏa mãn

1) Điều kiện về luật loại trừ (Mutual Exclusion)
   Tài nguyên phải là tài nguyên dành riêng không chia sẻ chung được. Do vậy nếu một tiến trình đang dùng tài nguyên thì tiến trình khác phải chờ tới lượt.

2) Điều kiện giữ và chờ (Hold and Wait)
   Một khi đã được cấp phát tài nguyên, tiến trình sẽ giữ tài nguyên đó và chờ tới khi được cấp phát nốt các tài nguyên còn lại.

3) Điều kiện không có quyền ưu tiên (no preemption)
   Các tiến trình không có quyền ưu tiên sử dụng tài nguyên. Tài nguyên đã được cấp cho tiến trình thì không thể bị giật ra để giao cho tiến trình khác.

4) Điều kiện chờ thành chu trình (circular wait)
   Việc phụ thuộc chờ tài nguyên lẫn nhau giữa các tiến trình tạo thành một vòng tròn, do vậy không tiến trình nào có đủ tài nguyên để hoàn tất công việc.

## Các biện pháp đương đầu với deadlock

Để đương đầu với deadlock chúng ta có các biện pháp sau

1) Phát hiện và khôi phục: Chấp nhận deadlock xảy ra, tìm cách phát hiện và khắc phục sự cố

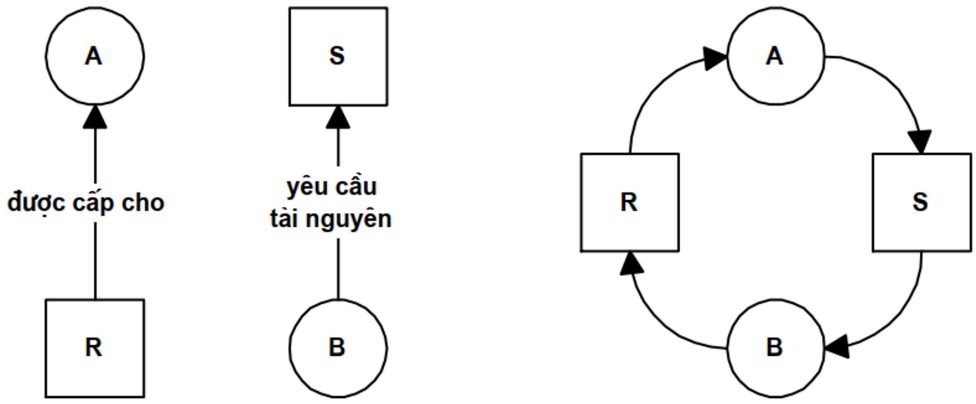
2) Ngăn ngừa deadlock bằng cách giám sát chặt chẽ thao tác cấp phát giải phóng tài nguyên

3) Ngăn ngừa deadlock bằng cách  bẻ gẫy một trong 4 điều kiện gây ra deadlock

4) Phương pháp đà điểu: Không làm gì cả. Xét rằng đối với các hệ thống thông thường thì phương pháp đà điểu không phải là tệ vì
- Xác suất xảy ra deadlock là thấp
- Nếu xảy ra thì hậu quả ít nghiêm trọng
- Chi phí đương đầu với deadlock lại cao

## Phát hiện và khôi phục

**Mô hình hóa đồ thị cấp phát tài nguyên**




Duy trì đồ thị cấp phát / giải phóng tài nguyên gồm 2 loại đỉnh: tiến trình và tài nguyên. Cạnh đồ thị biểu diễn tình trạng cấp phát tài nguyên cho tiến trình.

**Phát hiện và phá vỡ chu trình**

Phát hiện và phá vỡ chu trình bằng cách xóa bớt 1 đỉnh là tiến trình trong chu trình.
Nhận xét:
- Luôn phải cập nhật đồ thị và chạy thuật toán kiểm tra chu trình
- Việc xóa 1 tiến trình đang chạy có thể làm mất tính toàn vẹn của dữ liệu, không khôi phục được trạng thái trước đó

## Ngăn ngừa bằng cách giám sát cấp phát tài nguyên

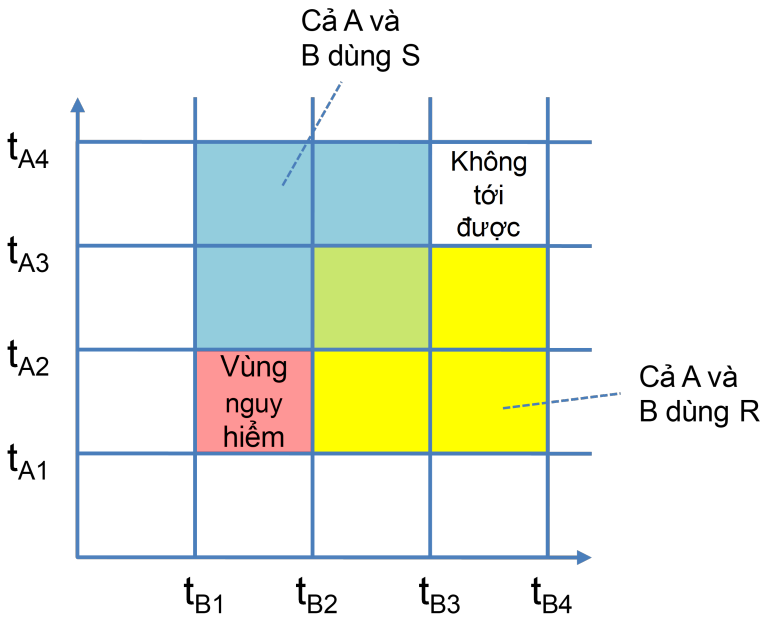
**Giám sát thông qua việc cấp phát CPU**

Biểu diễn đồ thị cấp phát CPU trong một hệ trục tọa độ mà mỗi trục là thời gian cấp phát CPU cho một tiến trình.
Đồ thị xuất phát từ gốc tọa độ và đi dọc theo các trục tọa độ, không được quay lui do thời gian không thể đảo ngược lại.
Các mốc cấp phát, giải phóng tài nguyên đối với mỗi tiến trình sẽ chia không gian thành các miền:

- Miền an toàn: tồn tại ít nhất một cách cấp phát CPU đáp ứng được mọi nhu cầu về tài nguyên của tiến trình
- Miền nguy hiểm: không có cách cấp phát CPU nào đáp ứng được mọi nhu cầu về tài nguyên của tiến trình
- Miền deadlock: thể hiện tình trạng bế tắc
- Miền không với tới được: đồ thị không thể đạt tới các trạng thái thuộc vùng này

Nhiệm vụ của hệ điều hành là "lái" đồ thị cấp phát CPU đi qua các vùng an toàn.

Minh họa với 2 tiến trình A và B. A xin cấp phát và giải phóng tài nguyên R vào thời điểm $t_{A1}$, $t_{A3}$; tài nguyên S vào thời điểm $t_{A2}$, $t_{A4}$. B xin cấp phát và giải phóng tài nguyên S vào thời điểm $t_{B1}$, $t_{B3}$; tài nguyên R vào thời điểm $t_{B2}$, $t_{B4}$.




Nhận xét: Phương pháp trực quan song không khả thi vì

- Việc xác định chính xác thời điểm sử dụng/giải phóng tài nguyên trước khi tiến trình chạy là không thực tế
- Số lượng các tiến trình luôn biến động nên phạm vi các miền cũng luôn thay

đổi

**Giám sát bằng thuật toán nhà băng với 1 loại tài nguyên**

Chúng ta mô phỏng việc giám sát tài nguyên thông qua hoạt động của một ngân hàng như sau

- Ngân hàng chỉ kinh doanh 1 loại tiền (1 loại tài nguyên)
- Danh sách khách hàng là cố định (không phát sinh tiến trình mới)
- Mỗi khách hàng phải thông báo trước tổng số tiền muốn vay (số tài nguyên sẽ dùng)
- Khi nhận đủ khoản vay, khách hàng sẽ hoàn tất công việc và hoàn trả số tiền vay ban đầu (trả lại tài nguyên nếu được cấp phát đủ)

Trạng thái của ngân hàng được định nghĩa là tình trạng cho vay hiện tại và số dư của ngân hàng, ví dụ

| | Có | Max |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

còn 3

Một trạng thái được gọi là an toàn nếu tồn tại ít nhất một cách cấp phát đáp ứng được tất cả các yêu cầu của khách hàng. Ngược lại là trạng thái không an toàn, sẽ dẫn tới deadlock.

Thuật toán nhà băng là kiểm tra trạng thái tiếp theo có an toàn không, nếu có thì phê duyệt yêu cầu cấp phát.

Ví dụ: A yêu cầu 1 tài nguyên. Nếu phê duyệt yêu cầu cấp phát này hệ thống sẽ chuyển sang trạng thái không an toàn và dẫn tới deadlock.

| | Có | Max |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

còn 3

| | Có | Max |
|---|---|---|
| A | 4 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |

còn 2

| | Có | Max |
|---|---|---|
| A | 4 | 9 |
| B | 4 | 4 |
| C | 2 | 7 |

còn 0

| | Có | Max |
|---|---|---|
| A | 4 | 9 |
| B | 0 | - |
| C | 2 | 7 |

còn 4

**Giám sát bằng thuật toán nhà băng với nhiều loại tài nguyên**

Giả sử chúng ta có $n$ tiến trình và $m$ tài nguyên mỗi loại. Gọi $\boldsymbol{Allocation}$ là ma trận tài nguyên đã cấp cho tiến trình, $\boldsymbol{Max}$ là ma trận tài nguyên tối đa cần bởi các tiến trình, cả 2 ma trận đều có cỡ $n \times m$. $Available$ là vector $m$ thành phần biểu diễn số tài nguyên mỗi loại còn dư.
Để thuận lợi, ta tính ma trận $\boldsymbol{Need} \coloneqq \boldsymbol{Max} - \boldsymbol{Allocation}$.

Trạng thái của ngân hàng được định nghĩa gồm ma trận $\boldsymbol{Allocation}$, $\boldsymbol{Need}$ và vector $Available$.

Thuật toán nhà băng

```
count=0;
do { // lặp cho tới khi tất cả tiến trình thỏa mãn hoặc không cấp phát được nữa
      found=FALSE;
      for (i=0; i<numProcesses; i++) {
            if ( satisfied[i]==FALSE && Need[i] <= Available) { // cấp phát được
                  available += Allocation[i];  // lấy lại tài nguyên đã cấp
                  satisfied[i]=TRUE;           // đánh dấu thỏa mãn
                  count++;                     // đếm số tiến trình thỏa mãn
                  found=TRUE;                  // vẫn đủ cấp phát
            }
      }
} while (count<numProcesses && found);

if (count<numProcesses) return FALSE;   // trạng thái nguy hiểm
else return TRUE;                       // trạng thái an toàn
```

Ví dụ 1: 5 tiến trình, 3 loại tài nguyên

Initial state

| Process | Allocation | Need | Satisfied |
|---|---|---|---|
| 0 | 0 2 0 | 7 2 3 | F |
| 1 | 1 0 0 | 1 2 3 | F |
| 2 | 2 0 1 | 5 1 1 | F |
| 3 | 1 1 2 | 1 1 1 | F |
| 4 | 0 0 2 | 4 2 1 | F |

Available = ( 3 1 1 )

Các bước cấp phát an toàn P3 -> P1 -> P2 -> P0 -> P4

Ví dụ 2: tiến trình, 3 loại tài nguyên

Initial state

| Process | Allocation | Need | Satisfied |
|---|---|---|---|
| 0 | 0 2 0 | 7 2 3 | F |
| 1 | 1 0 0 | 1 2 3 | F |
| 2 | 2 0 1 | 5 1 1 | F |
| 3 | 1 1 2 | 1 1 1 | F |
| 4 | 0 0 2 | 4 2 1 | F |

Available = ( 2 1 1 )

Không tồn tại cách cấp phát nào.