LAB 2.3: QUẢN LÝ TIẾN TRÌNH (DEADLOCK)

(2312678 – Nguyễn Lê Bảo Long – CTK47A)

# Mục tiêu

* + Các phương pháp giải quyết Deadlock
  + Giải thuật Banker

# Nội dung

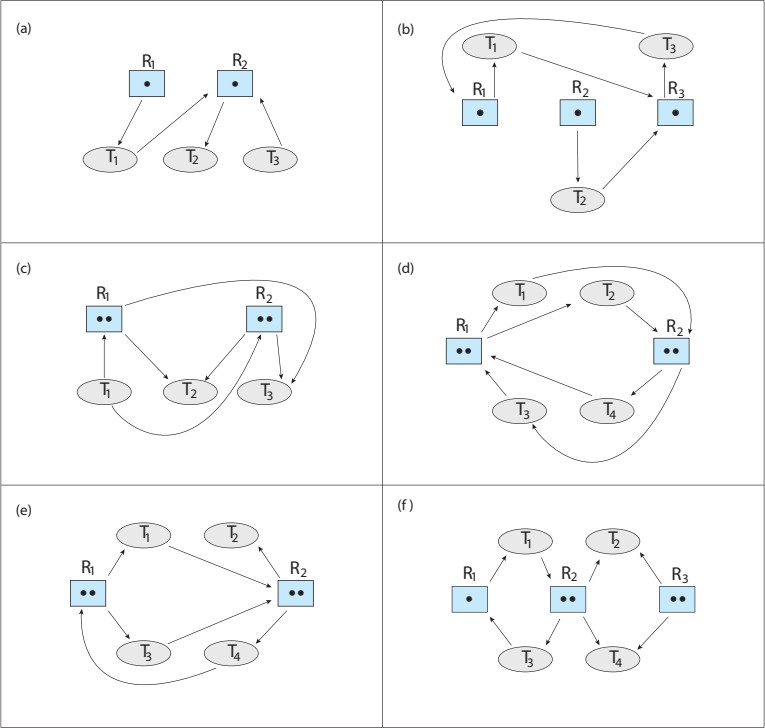
1. Có thể xảy ra deadlock nếu chỉ liên quan đến một tiến trình? Giải thích?

**Không thể xảy ra deadlock nếu chỉ có một tiến trình.** Lý do là deadlock yêu cầu ít nhất hai tiến trình hoặc nhiều hơn để tạo ra tình huống chờ đợi lẫn nhau (circular wait). Một tiến trình đơn lẻ không thể tự chờ đợi chính nó để giải phóng tài nguyên, vì nó không thể vừa giữ tài nguyên vừa chờ đợi tài nguyên khác mà không có sự tham gia của tiến trình khác.

**Ví dụ:**

* Nếu một tiến trình yêu cầu một tài nguyên và không thể nhận được nó (ví dụ, tài nguyên đã bị khóa bởi chính nó), nó sẽ bị block hoặc chờ đợi vô hạn, nhưng đây không phải là deadlock mà là một tình huống khác (ví dụ: livelock hoặc resource starvation).

1. Cho các đồ thị cấp phát tài nguyên (RAG) sau, đồ thị nào có deadlock? Với trường hợp có deadlock, cho biết chu trình giữa các thread và tài nguyên? Trường hợp không phải deadlock cho biết thứ tự các theard có thể hoàn thành?



Đồ thị (a):

* Không có deadlock.
* Lý do: Mỗi thread chỉ yêu cầu tối đa một tài nguyên tại một thời điểm và không có chu trình yêu cầu tài nguyên giữa các thread.

Đồ thị (b):

* Không có deadlock.
* Lý do: Tương tự như đồ thị (a), mỗi thread chỉ yêu cầu tối đa một tài nguyên tại một thời điểm và không có chu trình yêu cầu tài nguyên giữa các thread.

Đồ thị (c):

* Không có deadlock.
* Lý do: Tương tự như đồ thị (a) và (b), mỗi thread chỉ yêu cầu tối đa một tài nguyên tại một thời điểm và không có chu trình yêu cầu tài nguyên giữa các thread.

Đồ thị (d):

* Deadlock có thể xảy ra.
* Chu trình deadlock:
  + Thread T1 yêu cầu tài nguyên R1 và được cấp.
  + Thread T2 yêu cầu tài nguyên R2 và được cấp.
  + Thread T3 yêu cầu tài nguyên R1 và bị chặn vì R1 đang được T1 sử dụng.
  + Thread T1 yêu cầu tài nguyên R2 và bị chặn vì R2 đang được T2 sử dụng.
* Giải thích: Có chu trình yêu cầu tài nguyên giữa các thread T1, T2 và T3, dẫn đến deadlock.

Đồ thị (e):

* Không có deadlock.
* Lý do: Mỗi thread chỉ yêu cầu tối đa một tài nguyên tại một thời điểm và không có chu trình yêu cầu tài nguyên giữa các thread.

Đồ thị (f):

* Không có deadlock.
* Lý do: Tương tự như đồ thị (e), mỗi thread chỉ yêu cầu tối đa một tài nguyên tại một thời điểm và không có chu trình yêu cầu tài nguyên giữa các thread.

Thứ tự hoàn thành các thread:

Trong trường hợp không có deadlock, thứ tự hoàn thành các thread có thể thay đổi tùy thuộc vào thuật toán phân bổ tài nguyên được sử dụng. Tuy nhiên, một số ví dụ về thứ tự hoàn thành có thể bao gồm:

* Đồ thị (a): T1, T2
* Đồ thị (b): T1, T2, T3
* Đồ thị (c): T1, T2, T3
* Đồ thị (e): T1, T2, T3, T4
* Đồ thị (f): T1, T2, T3, T4

Lưu ý rằng đây chỉ là ví dụ và thứ tự thực tế có thể khác nhau.

Kết luận:

Dựa trên phân tích trên, only đồ thị (d) có khả năng xảy ra deadlock. Chu trình deadlock bao gồm các thread T1, T2 và T3. Trong các đồ thị còn lại, không có deadlock và thứ tự hoàn thành các thread có thể thay đổi tùy thuộc vào thuật toán phân bổ tài nguyên được sử dụng.

1. Xét hệ thống có 5 tiến trình và 4 loại tài nguyên với trạng thái sau

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Available | | | |
| A | B | C | D | A | B | C | D | A | B | C | D |
| T0 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 5 | 2 | 0 |
| T1 | 1 | 0 | 0 | 0 | 1 | 7 | 5 | 0 |  |  |  |  |
| T2 | 1 | 3 | 5 | 4 | 2 | 3 | 5 | 6 |  |  |  |  |
| T3 | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 |  |  |  |  |
| T4 | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 |  |  |  |  |

Sử dụng thuật giải Banker:

* 1. Tìm ma trận **Need** tương ứng?

Ta có:

+ Ma trận Allocation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Allocation | | | |
| A | B | C | D |
| T0 | 0 | 0 | 1 | 2 |
| T1 | 1 | 0 | 0 | 0 |
| T2 | 1 | 3 | 5 | 4 |
| T3 | 0 | 6 | 3 | 2 |
| T4 | 0 | 0 | 1 | 4 |

+ Ma trận Max

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Max | | | |
| A | B | C | D |
| T0 | 0 | 0 | 1 | 2 |
| T1 | 1 | 7 | 5 | 0 |
| T2 | 2 | 3 | 5 | 6 |
| T3 | 0 | 6 | 5 | 2 |
| T4 | 0 | 6 | 5 | 6 |

Ma trận Need được tính bằng công thức sau:

Need = Max – Allocation

* + Ma trận Need:
  1. Hệ thống có trong trạng thái an toàn? Giải thích?
* Một tiến trình có thể chạy được nếu Need <= Available
  + Need T0 = (0,0,0,0) < (1,5,2,0) => T0 có thể chạy.

Available mới = (1,5,2,0) + (0,0,1,2) = (1,5,3,2)

* + Need T2 = (1,0,0,2) < (1,5,3,2) => T2 có thể chạy

Avialable mới = (1,5,3,2) + (1,3,5,4) = (2,8,8,6)

* + Need T3 = (0,0,2,0) < (2,8,8,6) => T3 có thể chạy

Available mới = (2,8,8,6) + (0,6,3,2) = (2,14,11,8)

* + Need T4 = (0,6,4,2) < (2,14,11,8) => T4 có thể chạy

Available mới (2,14,11,8) + (0,0,1,4) = (2,14,12,12)

* + Need T1 = (0,7,5,0) < (2,14,12,12) => T1 có thể chạy

Available mới = (2,14,12,12) + (1,0,0,0) = (3,14,12,12)

* 1. Nếu **T1** yêu cầu tài nguyên (0,4,2,0) thì có được cấp phát?
  + Vì T1 (0, 4, 2, 0) <= Need (0, 7, 5, 0).

Và T1 (0,4, 2, 0) <= Available (1, 5, 2, 0).

* + Nên hệ thống có thể cấp phát tài nguyên cho T1 (0, 4, 2, 0).

1. Xét hệ thống có các tiến trình và tài nguyên với trình trạng sau:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | |
| A | B | C | D | A | B | C | D |
| T0 | 3 | 0 | 1 | 4 | 5 | 1 | 1 | 7 |
| T1 | 2 | 2 | 1 | 0 | 3 | 2 | 1 | 1 |
| T2 | 3 | 1 | 2 | 1 | 3 | 3 | 2 | 1 |
| T3 | 0 | 5 | 1 | 0 | 4 | 6 | 1 | 2 |
| T4 | 4 | 2 | 1 | 2 | 6 | 3 | 2 | 5 |

Sử dụng thuật giải Banker cho biết hệ thống có trong tình trạng an toàn không

trong các trường hợp sau. Nếu an toàn, cho biết chuỗi an toàn của các tiến trình.

Ma trận Need được tính bằng công thức sau:

Need = Max – Allocation

* + Ma trận Need:
  1. **Available**=(0,3,0,1)
* Need T2 (0, 2, 0, 0) <= Available (0, 3, 0, 1) => T2 có thể chạy.

Available mới = (3, 4, 2, 2)

* Need T1 (1, 0, 0, 1) <= Available (3, 4, 2, 2) => T1 có thể chạy.

Available mới (5, 6, 3, 2)

* Need T3 (4, 1, 0, 2) <= Available (5, 6, 3, 2) => T3 có thể chạy.

Available mới (5, 11, 4, 2)

* Need T0 (2, 1, 0, 3) >= Available (5, 11, 4, 2) => T0 không chạy được.
* Need T4 (2, 1, 1, 3) >= Available (5, 11, 4, 2) => T4 không chạy được
  + Hệ thống này có trong tình trạng không an toàn.
  1. **Available**=(1,0,0,2)
  + Need T1 (1, 0, 0, 1) <= Available (1, 0, 0, 2) => T1 có thể chạy.

Available mới (3, 2, 1, 2)

* + Need T2 (0, 2, 0, 0) <= Available (3, 2, 1, 2) => T2 có thể chạy.

Available mới (6, 3, 3, 3)

* + Need T0 (2, 1, 0, 3) <= Available (6, 3, 3, 3) => T0 có thể chạy.

Available mới (9, 3, 4, 7)

* + Need T3 (4, 1, 0, 2) <= Available (9, 3, 4, 7) => T3 có thể chạy.

Available mới (9, 8, 5, 7)

* + Need T4 (2, 1, 1, 3) <= Available (9, 8, 5, 7) => T4 có thể chạy.

Available mới (13, 10, 6, 9)

* + Hệ thống này có trong tình trạng không an toàn.

5) Xét hệ thống có các tiến trình và tài nguyên với trình trạng sau:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Available | | | |
| A | B | C | D | A | B | C | D | A | B | C | D |
| T0 | 3 | 1 | 4 | 1 | 6 | 4 | 7 | 3 | 2 | 2 | 2 | 4 |
| T1 | 2 | 1 | 0 | 2 | 4 | 2 | 3 | 2 |  |  |  |  |
| T2 | 2 | 4 | 1 | 3 | 2 | 5 | 3 | 3 |  |  |  |  |
| T3 | 4 | 1 | 1 | 0 | 6 | 3 | 3 | 2 |  |  |  |  |
| T4 | 2 | 2 | 2 | 1 | 5 | 6 | 7 | 5 |  |  |  |  |

. Dùng thuật giải Banker:

1. Cho biết chuỗi an toàn để các tiến trình có thể hoàn thành

Ma trận Need được tính bằng công thức sau:

Need = Max – Allocation

* + Ma trận Need:
* Need T2 (0, 1, 2, 0) <= Available (2, 2, 2, 4) => T2 có thể chạy.

Available mới (4, 6, 3, 7)

* Need T0 (3, 3, 3, 2) <= Available (4, 6, 3, 7) => T0 có thể chạy.

Available mới (7, 7, 7, 8)

* Need T1 (2, 1, 3, 0) <= Available (7, 7, 7, 8) => T1 có thể chạy.

Available mới (9, 8, 7, 10)

* Need T3 (2, 2, 2, 2) <= Available (9, 8, 7, 10) => T3 có thể chạy.

Available mới (13, 9, 8 ,10)

* + Need T4 (3, 4, 5, 4) <= Available (13, 9, 8, 10) => T4 có thể chạy.

Available mới (15, 11, 10, 11)

* Hệ thống an toàn.

1. Nếu T4 yêu cầu (2,2,2,4) thì có được cấp phát?

T4 (2, 2, 2, 4) <= Need T4 ( 3, 4, 5, 4)

T4 (2, 2, 2, 4) <= Available (2, 2, 2, 4)

* Hệ thống có thể cấp phát tài nguyên cho T4 (2, 2, 2, 4).

1. Nếu T2 yêu cầu (0,1,1,0) thì có được cấp phát?

T2 (0, 1, 1, 0) <= Need (0, 1, 2, 0)

T2 (0, 1, 1, 0) <= Available (2, 2, 2, 4)

* Hệ thống có thể cấp phát tài nguyên cho T2 (0, 1, 1, 0).

1. Nếu T3 yêu cầu (2,2,1,1) thì có được cấp phát?

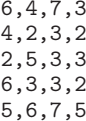
T3 (2, 2, 1, 1) <= Need (2, 2, 2, 2)

T3 (2, 2, 1, 1) <= Available (2, 2, 2, 4)

* Hệ thống có thể cấp phát tài nguyên cho T3 (2, 2, 1, 1).

1. Bài tập lập trình mô phỏng giải thuật Banker: Thiết kế chương trình cho phép user thực hiện các chức năng: Request (Yêu cầu), Release (Giải phóng) tài nguyên và hiển thị các thông tin các cấu trúc dữ liệu (Available, Maximum, Allocation và Need).

* Input: (có thể nhập hoặc đọc từ file)
  + Số loại tài nguyên và thể hiện tương ứng
  + Ma trận yêu cầu cấp phát tối đa
* Output: tùy vào lệnh chức năng thực hiện Ví dụ các file chứa thông tin khởi tạo:
* File tài nguyên có giá trị là **10 5 7 8**, cho biết có 4 loại tài nguyên với số thể hiện của mỗi loại lần lượt là 10, 5, 7 và 8
* File chứa thông tin yêu cầu cấp phát tối đa cho 5 tiến trình với từng loại tài nguyên có giá trị:



* Chương trình cho phép thực hiện các lệnh:
  + **RQ 0 3 1 2 1** : tiến trình thứ nhất yêu cầu các tài nguyên với số lượng lần lượt là 3, 1, 3, 1. Chương trình cho biết có cấp phát được không?
  + **RL 1 2 1 3 2**: tiến trình thứ hai giải phóng các tài nguyên tương ứng
  + \* : hiển thị thông tin của Available, Maximum, Allocation và Need

using System;

class BankerAlgorithm

{

private int[] Available;

private int[,] Maximum;

private int[,] Allocation;

private int[,] Need;

private int processCount;

private int resourceCount;

public BankerAlgorithm(int[] available, int[,] maximum, int[,] allocation)

{

Available = available;

Maximum = maximum;

Allocation = allocation;

processCount = Maximum.GetLength(0);

resourceCount = Available.Length;

// Tính toán Need = Maximum - Allocation

Need = new int[processCount, resourceCount];

for (int i = 0; i < processCount; i++)

{

for (int j = 0; j < resourceCount; j++)

{

Need[i, j] = Maximum[i, j] - Allocation[i, j];

}

}

}

// Kiểm tra có thể cấp phát tài nguyên cho tiến trình

public bool RequestResources(int process, int[] request)

{

// Kiểm tra yêu cầu tài nguyên hợp lệ

for (int i = 0; i < resourceCount; i++)

{

if (request[i] > Need[process, i] || request[i] > Available[i])

{

return false; // Yêu cầu không hợp lệ

}

}

// Cấp phát tài nguyên

for (int i = 0; i < resourceCount; i++)

{

Available[i] -= request[i];

Allocation[process, i] += request[i];

Need[process, i] -= request[i];

}

return true; // Cấp phát thành công

}

// Giải phóng tài nguyên

public void ReleaseResources(int process, int[] release)

{

for (int i = 0; i < resourceCount; i++)

{

Available[i] += release[i];

Allocation[process, i] -= release[i];

Need[process, i] += release[i];

}

}

// Hiển thị thông tin Available, Maximum, Allocation, Need

public void DisplayInfo()

{

Console.WriteLine("Available: " + string.Join(", ", Available));

Console.WriteLine("Maximum:");

for (int i = 0; i < processCount; i++)

{

for (int j = 0; j < resourceCount; j++)

Console.Write(Maximum[i, j] + " ");

Console.WriteLine();

}

Console.WriteLine("Allocation:");

for (int i = 0; i < processCount; i++)

{

for (int j = 0; j < resourceCount; j++)

Console.Write(Allocation[i, j] + " ");

Console.WriteLine();

}

Console.WriteLine("Need:");

for (int i = 0; i < processCount; i++)

{

for (int j = 0; j < resourceCount; j++)

Console.Write(Need[i, j] + " ");

Console.WriteLine();

}

}

}

class Program

{

static void Main()

{

// Ví dụ về tài nguyên và yêu cầu cấp phát

int[] available = { 10, 5, 7, 8 }; // Các tài nguyên có sẵn

int[,] maximum = {

{7, 5, 3, 2},

{3, 2, 2, 2},

{9, 0, 2, 4},

{2, 3, 3, 1},

{4, 3, 3, 2}

}; // Ma trận Maximum

int[,] allocation = {

{0, 1, 0, 0},

{2, 0, 0, 1},

{3, 2, 2, 2},

{2, 1, 1, 0},

{0, 0, 2, 1}

}; // Ma trận Allocation

BankerAlgorithm banker = new BankerAlgorithm(available, maximum, allocation);

// Hiển thị thông tin ban đầu

banker.DisplayInfo();

// Tiến trình yêu cầu tài nguyên

Console.WriteLine("\nRequest Resources (RQ 0 3 1 2 1):");

if (banker.RequestResources(0, new int[] { 3, 1, 3, 1 }))

{

Console.WriteLine("Resources allocated successfully.");

}

else

{

Console.WriteLine("Resources cannot be allocated.");

}

banker.DisplayInfo();

// Tiến trình giải phóng tài nguyên

Console.WriteLine("\nRelease Resources (RL 1 2 1 3 2):");

banker.ReleaseResources(1, new int[] { 2, 1, 3, 2 });

banker.DisplayInfo();

}

}