Thuật toán Banker

Hệ điều hành - OS (Trường Đại học Sài Gòn)

Tắc nghẽn (Deadlock)

Định nghĩa:

Một tập hợp các tiến trình được định nghĩa ở trong tình trạng tắc nghẽn khi mỗi tiến
trình trong tập hợp đều chờ đợi một sự kiện mà chỉ có một tiến trình khác trong tập hợp
mới có thể phát sinh được.

Nói cách khác, mỗi tiến trình trong tập hợp đều chờ được cấp phát một tài nguyên hiện
đang bị một tiến trình khác cũng ở trạng thái blocked chiếm giữ. Như vậy không có tiến
trình nào có thể tiếp tục xử lý , cũng như giải phóng tài nguyên cho tiến trình khác sử
dụng, tất cả các tiến trình trong tập hợp đều bị khóa vĩnh viễn !

Điều kiện xuất hiện tắc nghẽn

Điều kiện xuất hiện các tắc nghẽn Sau đây là các điều kiện cần có thể làm xuất hiện tắc nghẽn:

1) Khi sử dụng tài nguyên không thể chia sẻ
2) Sự chiếm giữ và yêu cầu thêm tài nguyên
3) Không thu hồi tài nguyên từ tiến trình đang giữ chúng
4) Tồn tại một chu kỳ trong đồ thị cấp phát tài nguyên.

Khi có đủ 4 điều kiện này, thì tắc nghẽn xảy ra. Nếu thiếu một trong 4 điều kiện trên thì không có tắc nghẽn.

Một số khái niệm cơ sở

- Trạng thái an toàn : trạng thái A là an toàn nếu hệ thống có thể thỏa mãn các nhu cầu tài nguyên (cho đến tối đa) của mỗi tiến trình theo một thứ tự nào đó mà vẫn ngăn chặn được tắc nghẽn.
- Một chuỗi cấp phát an toàn: một thứ tự của các tiến trình <P1, P2,...,Pn> là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình Pi nhu cầu tài nguyên của Pi có thể được thỏa mãn với các tài nguyên còn tự do của hệ thống, cộng với các tài nguyên đang bị chiếm giữ bởi các tiến trình Pj khác, với j<i.
- Một trạng thái an toàn không thể là trạng thái tắc nghẽn. Ngược lại một trạng thái không an toàn có thể dẫn đến tình trạng tắc nghẽn.
- Chiến lược cấp phát : chỉ thỏa mãn yêu cầu tài nguyên của tiến trình khi trạng thái kết quả là an toàn!

Giải thuật banker:

Ứng dụng: Dùng để xác định trạng thái an toàn.

Giải thuật xác định trạng thái an toàn:

Dạng 1:

Cần sử dụng các cấu trúc dữ liệu sau:

Int Available[numResources];

//Available[r] = số lượng các thể hiện còn tự do của tài nguyên r

Int Maxm[numProcs,NumResources]

//Maxm[p,r] = nhu cầu tối đa của tiến trình p về tài nguyên r

Int Allocation[numProcs,NumResources];

// Allocation[p,r] = số lượng tài nguyên r thực sự

// cấp phát cho p

Int Need[NumProcs,NumRources];

//Need[p,r] = Max[p,r] - Allocation[p,r]

Thuật toán dạng 1:

1. Khởi tạo :
   Work[NumProcs,NumResources] = Available
   Finish[NumProcs] = false;

2. Tìm i sao cho:
   A. Finish[i] == false;
   B. Need[i] <= Work[i]
   Nếu không có i như thế, đến bước 4.
3. Work = Work + Allocation[i];
   Finish[i] = true;
   Đến bước 2
4. Nếu Finish[i] == true với mọi i thì hệ thống an toàn.
   Nếu Finish[i] == false với 0 <= i <= n - 1 thì hệ thống đang bế tắc
   (và tiến trình Pi đang bế tắc)

Ví dụ: Hệ thống gồm:

- 5 tiến trình P0, P1, P2, P3, P4
- 3 loại tài nguyên A,B,C
+ A có 7 thể hiện
+ B có 2 thể hiện
+ C có 5 thể hiện

Giả sử trạng thái của hệ thống tại thời điểm T0:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3			
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			

Need		
A	B	C

Tiến trình	Work		
	A	B	C
	3	3	2

Need = Max - Allocation

If Need <= Work thì chúng ta cộng dồn Work = Work + Allocation

Need		
A	B	C
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Tiến trình	Work		
	A	B	C
	3	3	2
P1	5	3	2
P3	7	4	3
P4	7	4	5
P0	7	5	5
P2	10	5	7

Hệ thống an toàn vì tồn tại thứ tự an toàn:

P1, P3, P4, P0, P2

Dạng 2 (có bảng Request)

Cần sử dụng các cấu trúc dữ liệu sau:

Int Available[numResources];

//Available[r] = số lượng các thể hiện còn tự do của tài nguyên r

Int Maxm[numProcs,NumResources]

Int Allocation[numProcs,NumResources];

// Allocation[p,r] = số lượng tài nguyên r thực sự

// cấp phát cho p

//Request[p,r] = yêu cầu của tiến trình p về tài nguyên r

Thuật toán dạng 1:

1. Khởi tạo :
   Work[NumProcs,NumResources] = Available
   Finish[NumProcs] = false;

2. Tìm i sao cho:
   C. Finish[i] == false;
   D. Request[i] <= Work[i]
   Nếu không có i như thế, đến bước 4.
3. Work = Work + Allocation[i];
   Finish[i] = true;
   Đến bước 2

4. Nếu Finish[i] == true với mọi i thì hệ thống an toàn.
   Nếu Finish[i] == false với 0 <= i <= n - 1 thì hệ thống đang bế tắc (và tiến trình Pi đang bế tắc)

Ví dụ: Hệ thống gồm:

- 5 tiến trình P0, P1, P2, P3, P4
- 3 loại tài nguyên A,B,C
+ A có 7 thể hiện
+ B có 2 thể hiện
+ C có 6 thể hiện

Giả sử trạng thái của hệ thống tại thời điểm T0:

	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			

Tiến trình	Work		
	A	B	C
	0	0	0

Nếu Request <= Work thì Work = Work + Allocation

Tiến trình	Work		
	A	B	C
	0	0	0
P0	0	1	0
P2	3	1	3
P3	5	2	4
P4	5	2	6
P1	7	2	6

Hệ thống an toàn vì tồn tại thứ tự an toàn:
P0, P2, P3, P4, P1

Dạng 3:
Cần sử dụng các cấu trúc dữ liệu sau:
Int Available[numResources];
//Available[r] = số lượng các thể hiện còn tự do của tài nguyên r

Int Maxm[numProcs,NumResources]
//Maxm[p,r] = nhu cầu tối đa của tiến trình p về tài nguyên r

Int Allocation[numProcs,NumResources];
// Allocation[p,r] = số lượng tài nguyên r thực sự
// cấp phát cho p

Int Need[NumProcs,NumRources];
//Need[p,r] = Max[p,r] - Allocation[p,r]

Int request[numResources];
//Yêu cầu cấp phát

Thuật toán dạng 3: Tiến trình Pi yêu cầu request thể hiện của tài nguyên r

1. Khởi tạo :
   Work[NumProcs,NumResources] = Available
   Finish[NumProcs] = false;

2. Nếu request <= need[i] đến bước 3
   Ngược lại, xảy ra tình huống lỗi
3. Nếu request <= Available[i] đến bước 4
   Yêu cầu cấp phát thêm không thành công, thoát chương trình

4. Tìm i sao cho:
   E. Finish[i] == false;
   F. Need[i] <= Work[i]
   Nếu không có i như thế, đến bước 4.

5. Work = Work + Allocation[i];
   Finish[i] = true;
   Đến bước 2
6. Nếu Finish[i] == true với mọi i kết quả là an toàn,cấp phát thêm thành công
   Nếu Finish[i] == false với 0 <= i <= n - 1 thì hệ thống không cấp phát thêm được.

Ví dụ: Hệ thống gồm:
- 5 tiến trình P0, P1, P2, P3, P4
- 3 loại tài nguyên A,B,C
+ A có 7 thể hiện
+ B có 2 thể hiện
+ C có 6 thể hiện

Giả sử trạng thái của hệ thống tại thời điểm T0:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3			
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			

Giả sử quá trình P1 yêu cầu thêm (1,0,2) thể hiện. Cần cập nhật lại trạng thái mới thì xử lý thế nào ?

-Cập nhật lại Allocation của P1 là (3,0,2)

Điều kiện:

Request[1](1,0,2)<=Available (3,3,2) - >True

Request[1](1,0,2)<=Need[1] (1,2,2) - >True

- Nếu một trong hai điều kiện này sai thì kết luận việc yêu cầu cấp phát thêm là không thành công.
- Nếu cả hai điều kiện đều đúng thì tiến hành tính 2 ma trận Need và Work

-Tạo bảng Need: Giữ nguyên P0,P2,3,P4; chỉ cập nhật Need của P1

Need		
A	B	C
7	4	3
0	2	0
6	0	0
0	1	1
4	3	1

Tiến trình	Work		
	A	B	C
	2	3	0
P1	5	3	2
P3	7	4	3
P4	7	4	5
P0	7	5	5
P2	10	5	7

Tồn tại chuỗi an toàn P1, P3, P4, P0, P2 tức là yêu cầu cấp phát thêm của hệ thống thành công