TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

======***======

BÁO CÁO BÀI TẬP LỚN THUỘC HỌC PHẦN: NGUYÊN LÝ HỆ ĐIỀU HÀNH

LẬP TRÌNH MÔ PHỎNG CÁC PHƯƠNG PHÁP PHÁT HIỆN BẾ TẮC

GVHD:	Nguyễn Bá Nghiễn
Nhóm - Lớp:	Nhóm 9 – Lớp: IT6025.4
Thành viên:	Nguyễn Viết Khánh (2020604925)
	Đỗ Thành Đạt (2020603842)
	Nguyễn Văn Dũng (2020604280)
	Dương Trung Kiên (2020604160)
	Nguyễn Văn Tuấn (2020604351)

Hà Nội – Năm 2022

Mục Lục

Danh mục hình vẽ

Lời Nói Đầu

Trong môi trường đa chương, nhiều tiến trình có thể cạnh tranh một số giới hạn tài nguyên. Một tiến trình yêu cầu tài nguyên, nếu tài nguyên không sẵn dùng tại thời điểm đó, tiến trình đi vào trạng thái chờ. Tiến trình có thể không bao giờ chuyển trạng thái trở lại vì tài nguyên chúng yêu cầu đang bị giữ bởi những tiến trình đang chờ khác. Trường hợp này gọi là deadlock (bế tắc).

Trong giáo trình này, chúng ta sẽ mô tả các phương pháp mà hệ điều hành có thể sử dụng để kiểm tra xác định bế tắc. Vấn đề deadlock chỉ có thể trở thành vấn đề phổ biến, xu hướng hiện hành gồm số lượng lớn tiến trình, chương trình đa luồng, nhiều tài nguyên trong hệ thống và đặc biệt các tập tin có đời sống dài và những máy phục vụ cơ sở dữ liệu hơn là các hệ thống đóng.

CHƯƠNG 1: BẾ TẮC VÀ HIỆN TƯỢNG BẾ TẮC

1.1 Bế tắc và hiện tượng bế tắc

1.1.1. Bế tắc:

Bế tắc là hiện tượng: Trong hệ thống xuất hiện hai hay nhiều các tiến trình, mà các tiến trình trong tập này đều chờ đợi một sự kiện nào đó đang được một tiến trình trong tập này chiếm giữ. Và sự đợi này có thể kéo dài vô hạn nếu không có sự tác động từ bên ngoài.

Hình 1.1

1.1.2. Hiện tượng bế tắc:

– Bế tắc trong máy tính:

– Giả sử có hai tiến trình P1 và P2 hoạt động đồng thời trong hệ thống. Tiến trình P1 đang giữ tài nguyên R1 và xin được cấp R2 để tiếp tục hoạt động, trong khi đó tiến trình P2 đang giữ tài nguyên R2 và xin được cấp R1 để tiếp tục hoạt động. Trong trường hợp này cả P1 và P2 sẽ không tiếp tục hoạt động được. Như vậy P1 và P2 rơi vào trạng thái tắc nghẽn.

•Tiến trình P1:
{
…
Khóa file R1;
...
Mở file R2;
…
Đóng R1 (mở khóa R1);
 	
}
	•Tiến trình P2
	{
	…
	Khóa file R2;
	...
	Mở file R1;
	…
	Đóng R1 (mở khóa R1);
	
	}

Trong trường hợp ở trên: hai tiến trình P1 và P2 sẽ rơi vào trạng thái tắc nghẽn, nếu không có sự can thiệp của hệ điều hành. Để phá bỏ tắc nghẽn này hệ điều hành có thể cho tạm dừng tiến trình P1 để thu hồi lại tài nguyên R1, lấy R1 cấp cho tiến trình P2 để P2 hoạt động và kết thúc, sau đó thu hồi cả R1 và R2 từ tiến trình P2 để cấp cho P1 và tái kích hoạt P1 để P1 hoạt động trở lại. Như vậy sau một khoảng thời gian cả P1 và P2 đều ra khỏi tình trạng tắc nghẽn.

Khi hệ thống xảy ra tắc nghẽn nếu hệ điều hành không kịp thời phá bỏ tắc nghẽn thì hệ thống có thể rơi vào tình trạng treo toàn bộ hệ thống. Như trong trường hợp tắc nghẽn ở ví dụ 1, nếu sau đó có tiến trình P3, đang giữ tài nguyên R3, cần R2 để tiếp tục thì P3 cũng sẽ rơi vào tập tiến trình bị tắc nghẽn, rồi sau đó nếu có tiến trình P4 cần tài nguyên R1 và R3 để tiếp tục thì P4 cũng rơi vào tập các tiến trình bị tắc nghẽn như P3, … cứ thế dần dần có thể dẫn đến một thời điểm tất cả các tiến trình trong hệ thống đều rơi vào tập tiến trình tắc nghẽn. Và như vậy hệ thống sẽ bị treo hoàn toàn.

1.2. Điều kiện xảy ra bế tắc trong hệ thống:

Hiện tượng bế tắc xảy ra khi và chỉ khi trong hệ thống tồn tại bốn điều kiện sau:

– Có tài nguyên găng.

– Quá trình phải đang giữ ít nhất một tài nguyên và đang chờ để nhận tài nguyên thêm mà hiện đang được giữ bởi quá trình khác.

– Không có hệ thống phân phối lại tài nguyên, việc sử dụng tài nguyên không bị ngắt.

– Có hiện tượng chờ đợi vòng tròn xảy ra.

Một tập hợp các quá trình {P0, P1, …, Pn} đang chờ mà trong đó P0 đang chờ một tài nguyên được giữ bởi P1, P1 đang chờ tài nguyên đang giữ bởi

P2, …, Pn-1 đang chờ tài nguyên đang được giữ bởi quá trình P0.

Lưu ý: tất cả bốn điều kiện phải cùng phát sinh để hiện tượng bế tắc xảy ra. Điều kiện chờ đợi chu trình đưa đến điều kiện giữ-và-chờ vì thế bốn điều kiện không hoàn toàn độc lập.

1.3. Các mức phòng tránh bế tắc:

Để tránh được bế tắc thông thường hệ thống áp dụng 3 mức:

– Ngăn ngừa: áp dụng các biện pháp để hệ không rơi vào bế tắc.

– Dự báo và tránh bế tắc: áp dụng các biện pháp kiểm tra xem tiến trình có bị rơi vào trạng thái bế tặc hay không. Nếu có thì thông báo trước khi bế tắc xảy ra.

– Nhận biết và khắc phục: tìm cách khắc phục và giải quyết.

CHƯƠNG 2: XÁC ĐỊNH BẾ TẮC

2.1. Xử lý deadlock:

Khi hệ thống gặp bế tắc hệ điều hành có thể sử dụng phương pháp sau:

– Thông báo cho Operator biết để xử lý.

– Đình chỉ hoạt động của tiến trình: phương pháp này dựa trên việc thu hồi lại các tài nguyên của những tiến trình bị kết thúc. Có thể sử dụng một trong hai cách đình chỉ sau:

+ Đình chỉ hoạt động của mọi tiến trình trong tình trạng bế tắc.

+ Đình chỉ hoạt động lần lượt của từng tiến trình cho tới khi thoát khỏi tình trạng bế tắc (khi đình chỉ tiến trình nào thì thu hồi lại tài nguyên của tiến trình đó).

Chú ý: khi đình chỉ hoạt động của các tiến trình cần chú ý tới các yếu tố sau:

+ Độ ưu tiên của tiến trình.

+ Tiến trình đã diễn ra bao lâu và còn bao lâu thì hoàn thành.

+ Có bao nhiêu kiểu tài nguyên và số lượng tài nguyên mà tiến trình đã dùng.

+ Tiến trình còn cần bao nhiêu tài nguyên để hoàn thành một công việc.

– Thu hồi tài nguyên: Áp dụng biện pháp ngắt tài nguyên từ một số tiến trình để cấp phát cho các tiến trình đang có nhu cầu sau đó kiểm tra lại tình trạng bế tắc. Phương pháp này cần phải:

+ Xem xét và lựa chọn tiến trình nào để ngắt tài nguyên và ngắt tài nguyên nào?

+ Khả năng phục hồi trạng thái ban đầu của tiến trình có thực hiện được hay không?

+ Có thể xảy ra khả năng một số tiến trình không bao giờ được cấp đủ tài nguyên.

2.2. Cách giải quyết deadlock (bế tắc):

Phần lớn, chúng ta có thể giải quyết vấn đề deadlock theo một trong ba cách:

- Chúng ta có thể sử dụng một giao thức để ngăn chặn hay tránh deadlocks, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock.
- Chúng ta có thể cho phép hệ thống đi vào trạng thái deadlock, phát hiện nó và phục hồi.
- Chúng ta có thể bỏ qua hoàn toàn vấn đề này và giả vờ deadlock không bao giờ xảy ra trong hệ thống. Giải pháp này được dùng trong nhiều hệ điều hành, kể cả UNIX.

Để đảm bảo deadlock không bao giờ xảy ra, hệ thống có thể dùng kế hoạch ngăn chặn hay tránh deadlock. Ngăn chặn deadlock là một tập hợp các phương pháp để đảm bảo rằng ít nhất một điều kiện cần (trong phần I) không thể xảy ra.

Ngược lại, tránh deadlock yêu cầu hệ điều hành cung cấp những thông tin bổ sung tập trung vào loại tài nguyên nào một quá trình sẽ yêu cầu và sử dụng trong thời gian sống của nó. Với những kiến thức bổ sung này, chúng ta có thể quyết định đối với mỗi yêu cầu quá trình nên chờ hay không. Để quyết định yêu cầu hiện tại có thể được thoả mãn hay phải bị trì hoãn, hệ thống phải xem xét tài nguyên hiện có, tài nguyên hiện cấp phát cho mỗi quá trình, và các yêu cầu và giải phóng tương lai của mỗi quá trình.

Nếu một hệ thống không dùng giải thuật ngăn chặn hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống có thể cung cấp một giải thuật để xem xét trạng thái của hệ thống để xác định deadlock có xảy ra hay không và giải thuật phục hồi từ deadlock.

Nếu hệ thống không đảm bảo rằng deadlock sẽ không bao giờ xảy ra và cũng không cung cấp một cơ chế để phát hiện và phục hồi deadlock thì có thể dẫn đến trường hợp hệ thống ở trong trạng thái deadlock. Trong trường hợp này, deadlock không được phát hiện sẽ làm giảm năng lực hệ thống vì tài nguyên đang được giữ bởi những quá trình mà chúng không thể thực thi, đi vào trạng thái deadlock. Cuối cùng, hệ thống sẽ dừng các chức năng và cần được khởi động lại bằng thủ công.

Mặc dù phương pháp này dường như không là tiếp cận khả thi đối với vấn đề deadlock nhưng nó được dùng trong một số hệ điều hành. Trong nhiều hệ thống, deadlock xảy ra không thường xuyên; do đó phương pháp này là rẻ hơn chi phí cho phương pháp ngăn chặn deadlock, tránh deadlock, hay phát hiện và phục hồi deadlock mà chúng phải được sử dụng liên tục.

Trong một số trường hợp, hệ thống ở trong trạng thái cô đặc nhưng không ở trạng thái deadlock. Như thí dụ, xem xét một quá trình thời thực chạy tại độ ưu tiên cao nhất (hay bất cứ quá trình đang chạy trên bộ định thời biểu không trưng dụng) và không bao giờ trả về điều khiển đối với hệ điều hành. Do đó, hệ thống phải có phương pháp phục hồi bằng thủ công cho các điều kiện không deadlock và có thể đơn giản sử dụng các kỹ thuật đó cho việc phục hồi deadlock.

2.3. Thuật toán phát hiện bế tắc:

+ Mỗi nguồn tài nguyên có thể có nhiều hơn một đơn vị

n = số tiến trình, m = số nguồn tài nguyên .

+ Available: mảng gồm m phần tử. Nếu Available [j] = k, thì có k đơn vị tài nguyên Rj có sẵn.

+ Allocation: Ma trận n x m. Nếu Allocation[i,j] = k thì Pi , đang chiêm giữ k đơn vị của Rj.

+ Request: Ma trận n x m. Nếu Request[i,j] = k, thì Pi , đang yêu cầu thêm k đơn vị của Rj.

1. Work và Finish là hai mảng có độ dài m và n. Khởi tạo:

(a) Work = Available

(b) Với i = 0,1,……,n-1, nếu Allocation[i] # 0 thì Finish [i] = false, ngược lại Finish [i] = True .

2. Tìm I sao cho thỏa mãn

(a) Finish [i] = false

(b) Requesti ≤ Work

Nếu không có i thỏa mãn, chuyển sang bước 4.

3. Work = Work + Allocation[i]

Finish [i] = true

Chuyển sang bước 2 .

4. Nếu Finish [i] == false với một số 0 ≤ i ≤ n – 1, thì hệ thống ở trạng thái bế tắc.

Và nếu Finish [i] == false thì Pi bị bế tắc.

** Ví dụ thuật toán phát hiện bế tắc.*

+ 5 tiến trình P0 P1 P2 P3 P4

+ 3 nguồn tài nguyên : A(7 đơn vị), B(2 đơn vị) and C(6 đơn vị)

+ Tại thời điểm t0, trạng thái của hệ thống là:

	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	

+ Chuỗi<P0, P2, P3, P1 ,P4 > cho phép Finish [i] = true với tất cả 0 ≤ i ≤ n-1.

+ P2 yêu cầu thêm 1 đơn vị tài nguyên C.

	Request		
	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

** Tần suất sử dụng thuật toán nhận diện.*

Phú thuộc vào :

+ Tần suất bế tắc xuất hiện.

+ Số lượng tiến trình liên quan khi bế tắc xảy ra.

2.4. Lập trình mô phỏng phương pháp kiểm tra xác định bế tắc

Giả sử có một dãy n tiến trình với m kiểu tài nguyên.

Allocation: là một mảng mxn thể hiện số tài nguyên mà mỗi kiểu hiện đã phân bổ cho các tiến trình.

Available: là mảng 1xm thể hiện số tài nguyên thể hiện số tài nguyên có thể sử dụng của mỗi kiểu.

Max: là mảng mxn thể hiện số tài nguyên cực đại mà mỗi tiến trình yêu cầu
Need: là mảng nxm thể hiện số tài nguyên còn cần của mỗi tiến trình

Request: là mảng nxm thể hiện số yêu cầu tài nguyên của mỗi tiến trình tại mỗi thời điểm.

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
  {
  int num_p,num_re;
  int all[10] [10],re[10],av[10],cl[10][ 10],used[ 10];
  int more[10];
  int i=0;
  int ap,ar;
  int cp,cr;
  int j=0;
  int pi,ci;
  printf("\nNhap so tien trinh : ");
```

```
scanf("%d",&num_p) ;
printf("\nNhap so tai nguyen : ");
scanf("%d",&num_re);
printf("\nNhap gia tri toi da cua moi tai nguyen : \n\n");
for(i=0;i<num_re;i++)
    {
    printf("R[%d] = ",i+1);
    scanf("%d",&re[i]) ;
    }

//Allocation
printf("\n");
for(ap=0;ap<num_p;ap++)
    {
    printf("Nhap so tai nguyen da cap phat cho tien trinh %d : ",ap+1);
    for(ar=0;ar<num_re;ar++)
    scanf("%d",&all[ar][ap]);
    }

//claim matrix
printf("\n");
for(cp=0;cp<num_p;cp++)
    {
    printf("Nhap so tai nguyen yeu cau cua tien trinh %d : ",cp+1);
```

```
    for(cr=0;cr<num_re;cr++)
    scanf("%d",&cl[cr][cp] );
    }

//Used & Available
for(i=0;i<num_re;i++)
    {
    used[i]=0;
    for(j=0;j<num_p;j++)
    used[i]=used[i]+all[i] [j];
    av[i]=re[i]-used[ i];
    }

//check deadlock
for(pi=0;pi<num_p;pi++)
    {
    int check=0;
    for(ci=0;ci<num_re;ci++)
        {
        more[ci]=cl[ci][pi]- all[ci][pi];
        if(more[ci] <= av[ci])
        check++;
        }
    if(check==num_re)
```

```
    goto l1;
    }
printf("\n\t\t!! XAY RA DEADLOCK !!\n\n");
getch();
return 0;

l1:
printf("\n\t\t!! SAFE !!\n\n");
getch();
return 0;
}
```

CHƯƠNG 3: BÀI TẬP ỨNG DỤNG

Xét một hệ thống với 5 quá trình từ P0 tới P4, và 3 loại tài nguyên A, B, C.

Loại tài nguyên A có 10 thể hiện, loại tài nguyên B có 5 thể hiện và loại tài nguyên C có 7 thể hiện. Giả sử rằng tại thời điểm T0 trạng thái hiện tại của hệ thống như sau:

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
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			

Nội dung ma trận Need được định nghĩa là Max-Allocation và là:

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	2
P3	0	1	1
P4	4	3	1

Chúng ta khẳng định rằng hệ thống hiện ở trong trạng thái an toàn. Thật vậy,

thứ tự<P1, P3, P4, P2, P0> thỏa tiêu chuẩn an toàn. Giả sử bây giờ P1 yêu cầu thêm

một thể hiện loại A và hai thể hiện loại C, vì thế Request 1= (1, 0, 2). Để quyết định

yêu cầu này có thể được cấp tức thì hay không, trước tiên chúng ta phải kiểm tra

Request 1 ≤ Available (nghĩa là, (1, 0, 2)) ≤ (3, 3, 2)) là đúng hay không. Sau đó, chúng ta giả sử yêu cầu này đạt được và chúng ta đi đến trạng thái mới sau :

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0
P1	3	0	2	0	2	0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

Chúng ta phải xác định trạng thái mới này là an toàn hay không. Để thực hiện điều này, chúng ta thực thi giải thuật an toàn của chúng ta và tìm thứ tự

<P1, P3, P4, P0, P2> thỏa yêu cầu an toàn. Do đó, chúng ta có thể cấp lập tức yêu cầu của quá trình P1.

Tuy nhiên, chúng ta cũng thấy rằng, khi hệ thống ở trong trạng thái này, một

yêu cầu (3, 3, 0) bởi P4 không thể được gán vì các tài nguyên là không sẵn dùng. Một yêu cầu cho (0, 2, 0) bởi P0 không thể được cấp mặc dù tài nguyên là sẵn dùng vì trạng thái kết quả là không an toàn.

CHƯƠNG 4: KẾT LUẬN

Nếu một hệ thống không dùng giải thuật ngăn chặn hay tránh deadlock thì trường hợp deadlock có thể xảy ra. Trong môi trường này, hệ thống có thể cung cấp một giải thuật để xem xét trạng thái của hệ thống để xác định deadlock có xảy ra hay không và giải thuật phục hồi từ deadlock.

Nếu hệ thống không đảm bảo rằng deadlock sẽ không bao giờ xảy ra và cũng không cung cấp một cơ chế để phát hiện và phục hồi deadlock thì có thể dẫn đến trường hợp hệ thống ở trong trạng thái deadlock. Trong trường hợp này, deadlock không được phát hiện sẽ làm giảm năng lực hệ thống vì tài nguyên đang được giữ bởi những quá trình mà chúng không thể thực thi, đi vào trạng thái deadlock. Cuối cùng, hệ thống sẽ dừng các chức năng và cần được khởi động lại bằng thủ công.

Mặc dù phương pháp này dường như không là tiếp cận khả thi đối với vấn đề deadlock nhưng nó được dùng trong một số hệ điều hành. Trong nhiều hệ thống, deadlock xảy ra không thường xuyên; do đó phương pháp này là rẻ hơn chi phí cho phương pháp ngăn chặn deadlock, tránh deadlock, hay phát hiện và phục hồi deadlock mà chúng phải được sử dụng liên tục.

TÀI LIỆU THAM KHẢO

- Nguyễn Thanh Hải, Giáo trình Nguyên lý hệ điều hành, 2016.
- Abraham Silberschatz, Galvin, Gagne, Operating System Concepts 8th edition.
-https://tailieumienphi.vn/doc/giao-trinh-nguyen-ly-he-dieu-hanh-dang-vu-tungwj2ttq.html?fbclid=IwAR1EtRutlXY2xvGNQhdPsDE1W83J1POIprhlp3rBW6z64MvMNPottgwvYT4
-https://tailieuvnu.com/giao-trinh-nguyen-ly-he-dieu-hanh-ho-dac-phuong/?fbclid=IwAR0JqpCUky-W01ZtTqG5oMC6kBR2haURG-uJgnlYJI9d-pyXTQ8cWnzogJ8