

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN MÔN HỌC: HỆ ĐIỀU HÀNH
ĐỀ TÀI: LẬP TRÌNH MÔ PHỎNG CÁC PHƯƠNG PHÁP
KIỂM TRA TÍNH AN TOÀN CỦA HỆ

Giáo viên:	Ths Nguyễn Bá Nghiễn
Nhóm số:	10
Lớp:	20212IT6025004 – K15

Hà Nội, 2022

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN MÔN HỌC: HỆ ĐIỀU HÀNH
ĐỀ TÀI: LẬP TRÌNH MÔ PHỎNG CÁC PHƯƠNG PHÁP
KIỂM TRA TÍNH AN TOÀN CỦA HỆ

Giáo viên: Ths Nguyễn Bá Nghiễn
Nhóm số: 10
Sinh viên thực hiện: Hoàng Trung Hiếu - 2020602568
Nguyễn Duy Hùng - 2020600796
Đào Quang Minh - 2019605021
Trần Anh Tú - 2020606616
Lớp: 20212IT6025004 – K15

Hà Nội, 2022

MỤC LỤC

LỜI MỞ ĐẦU	4
Giới thiệu chung, khái quát về đề tài được giao	5
Chương 1: Sơ lược về an toàn của hệ thống.....	5
1.1 Bế tắc (Deadlock)	5
<i>Hình 1.1.1 Minh họa 3 Process thực thi</i>	<i>6</i>
1.2 Ngăn chặn bế tắc và an toàn hệ thống.....	7
1.2.1 Ngăn chặn bế tắc.....	7
1.2.2 Khái niệm dãy tiến trình an toàn	9
Chương 2: Thuật toán kiểm tra tính an toàn hệ thống	9
2.1 . Thuật toán chuyển sang trạng thái an toàn	9
2.2 . Thuật toán kiểm tra tính an toàn của hệ	10
2.4 . Bài code của thuật toán	11
2.5 . Ví dụ minh họa cho code	14
<i>Hình 2.5.1 Nhập số tiến trình</i>	<i>15</i>
<i>Hình 2.5.2 Nhập số tài kiểu tài nguyên</i>	<i>16</i>
<i>Hình 2.5.3 Nhập tài nguyên của mỗi kiểu (Available)</i>	<i>16</i>
<i>Hình 2.5.4 Nhập tài nguyên của mỗi kiểu đã phân bổ (Allocation)</i>	<i>17</i>
<i>Hình 2.5.5 Nhập số tài nguyên cực đại (Max)</i>	<i>17</i>
<i>Hình 2.5.6 Hiện thị kết quả trên màn hình Console</i>	<i>18</i>
2.6 . Ví dụ làm ở dạng lý thuyết	18
KẾT LUẬN.....	21
TÀI LIỆU THAM KHẢO.....	22

LỜI MỞ ĐẦU

Môn học “Nguyên lý hệ điều hành” cung cấp cho người học những kiến thức chung nhất về hệ điều hành máy tính, giúp người học nắm bắt được những nguyên lý cơ bản và nguyên tắc làm việc của một hệ điều hành máy tính tổng quát. Từ đó, áp dụng để làm việc với các hệ điều hành cụ thể trên thực tế, hiểu và xử lý được các vấn đề có thể xảy ra trong hệ thống. Đồng thời, nắm bắt được xu hướng phát triển của các hệ điều hành trong tương lai.

Báo cáo “Bài tập lớn môn hệ điều hành” của nhóm 10 với đề tài “Lập lịch mô phỏng các phương pháp kiểm tra tính an toàn của hệ” gồm có các nội dung chính như sau:

Phần đầu là giới thiệu chung về đề tài được giao. Phần thứ hai là sơ lược về an toàn của hệ thống. Phần thứ ba trình bày về thuật toán kiểm tra tính an toàn của hệ. Phần thứ tư trình bày về Phát hiện và xử lý bế tắc. Phần thứ năm đưa ra một ví dụ minh họa cho bài toán. Cuối cùng là phần tài liệu mà nhóm dùng để tham khảo.

Báo cáo bài tập lớn của chúng em còn nhiều thiếu sót, chúng em rất mong nhận được những nhận xét và sự giúp đỡ của thầy.

Chúng em xin chân thành cảm ơn!

Giới thiệu chung, khái quát về đề tài được giao

Hệ điều hành muốn hoạt động tốt thì cần phải có các thuật toán dùng để kiểm tra, phát hiện và xử lý các vấn đề thường xuyên xảy ra đối với hoạt động của hệ điều hành. Chẳng hạn như bế tắc hay việc phân bổ tài nguyên sao cho tránh lãng phí và đạt hiệu quả sử dụng tối đa.

Chính vì thế mà các thuật toán dùng để kiểm tra tính an toàn của hệ được phát triển bằng ngôn ngữ C++.

Chương 1: Sơ lược về an toàn của hệ thống

1.1 Bế tắc (Deadlock)

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

Bế tắc là tình huống xuất hiện khi hai tiến trình phải chờ đợi nhau giải phóng tài nguyên hoặc nhiều tiến trình chờ sử dụng các tài nguyên theo một “vòng tròn” (circular chain)

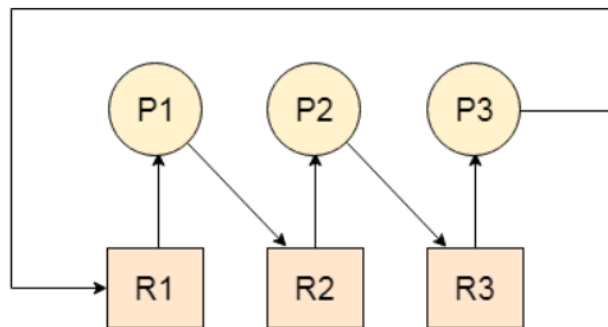
Hầu hết các hệ điều hành không cung cấp phương tiện ngăn chặn deadlock. Vấn đề deadlock chỉ trở thành vấn đề phổ biến, xu hướng hiện hành gồm số lượng lớn tiến trình, chương trình đa luồng, nhiều tài nguyên trên hệ thống và đặc biệt các tập tin có đời sống dài và những máy phục vụ cơ sở dữ liệu hơn là hệ thống đóng.

Giả sử rằng có ba Process P1, P2 và P3. Có ba tài nguyên khác nhau R1, R2 và R3. R1 được gán cho P1, R2 được gán cho P2 và R3 được gán cho P3.

Sau một thời gian, P1 yêu cầu R1 đang được P2 sử dụng. P1 tạm dừng Process thực thi của nó vì nó không thể hoàn thành nếu không có R2. P2 cũng yêu cầu R3 đang được P3 sử dụng. P2 cũng dừng Process thực thi của nó vì nó

không thể tiếp tục mà không có R3. P3 cũng yêu cầu R1 đang được P1 sử dụng do đó P3 cũng dừng thực thi.

Trong kịch bản này, một chu trình đang được hình thành giữa ba Process. Không có Process nào đang diễn ra và tất cả đều đang chờ đợi. Máy tính không phản hồi vì tất cả các Process đã bị chặn.



Hình 1.1.1 Minh họa 3 Process thực thi

Điều kiện để xảy ra bế tắc nếu 4 điều kiện sau xuất hiện đồng thời (điều kiện cần)

- Loại trừ lẫn nhau (mutual exclusion)
 - + Một tài nguyên bị chiếm bởi một tiến trình, và không tiến trình nào khác có thể sử dụng tài nguyên này
- Giữ và chờ (hold and wait)
 - + Một tiến trình giữ ít nhất một tài nguyên và chờ một số tài nguyên khác rồi để sử dụng. Các tài nguyên này đang bị một tiến trình khác chiếm giữ
- Không có đặc quyền (no preemption)
 - + Tài nguyên bị chiếm giữ chỉ có thể rời khi tiến trình “tự nguyện” giải phóng tài nguyên sau khi đã sử dụng xong.
- Chờ vòng (circular wait)
 - + Một tập tiến trình $\{P_0, P_1, \dots, P_n\}$ có xuất hiện điều kiện “chờ vòng” nếu P_0 chờ một tài nguyên do P_1 chiếm giữ, P_1 chờ một tài nguyên khác do P_2 chiếm giữ, ..., P_{n-1} chờ tài nguyên do P_n chiếm giữ và P_n chờ tài nguyên do P_0 chiếm giữ

Phòng tránh bế tắc:

- Ngăn ngừa: Áp dụng các biện pháp để hệ thống không rơi vào trạng thái bế tắc.
- Dự báo và tránh bế tắc: Áp dụng các biện pháp kiểm tra xem tiến trình có rơi vào bế tắc hay không, nếu có thông báo trước khi xảy ra.
- Nhận biết và khắc phục: tìm cách phát hiện và giải quyết.

1.2 Ngăn chặn bế tắc và an toàn hệ thống

1.2.1 Ngăn chặn bế tắc

Mỗi khi phân bổ tài nguyên cho các tiến trình, hệ thống sẽ kiểm tra xem liệu việc phân bổ đó có đẩy hệ thống vào tình trạng bế tắc hay không. Nếu có tìm cách giải quyết trước khi bế tắc xảy ra.

Ngăn chặn bế tắc (deadlock prevention) là phương pháp xử lý bế tắc, không cho nó xảy ra bằng cách làm cho ít nhất một điều kiện cần của bế tắc là loại trừ lẫn nhau, giữ và chờ, không có đặc quyền hoặc chờ vòng không được thỏa mãn (không xảy ra).

Để ngăn chặn bế tắc cần đảm bảo sao cho 4 điều kiện xảy ra bế tắc không xảy ra đồng thời.

- Ngăn chặn “Loại trừ lẫn nhau”:

- + Loại trừ lẫn nhau: là điều kiện bắt buộc cho các tài nguyên không sử dụng chung được → Khó làm cho C1 không xảy ra vì các hệ thống luôn có các tài nguyên không thể sử dụng chung được.

- Ngăn chặn “Giữ và chờ”: Có thể làm cho “Giữ và chờ” không xảy ra bằng cách đảm bảo:

- + Một tiến trình luôn yêu cầu cấp phát tài nguyên chỉ khi nó không chiếm giữ bất kỳ một tài nguyên nào, hoặc
- + Một tiến trình chỉ thực hiện khi nó được cấp phát toàn bộ các tài nguyên cần thiết

- Ngăn chặn “không có đặc quyền”: Để ngăn chặn không cho điều kiện này xảy ra, có thể sử dụng giao thức sau:

- + Nếu tiến trình P (đang chiếm tài nguyên R_1, \dots, R_{n-1}) yêu cầu cấp phát tài nguyên R_n nhưng không được cấp phát ngay (có nghĩa là P phải chờ) thì tất cả các tài nguyên R_1, \dots, R_{n-1} phải được “thu hồi”
- + Nói cách khác, R_1, \dots, R_{n-1} phải được “giải phóng” một cách áp đặt, tức là các tài nguyên này phải được đưa vào danh sách các tài nguyên mà P đang chờ cấp phát.

- Mã lệnh ngăn chặn “không có đặc quyền”:

- + Tiến trình P yêu cầu cấp phát tài nguyên R_1, \dots, R_{n-1} if (R_1, \dots, R_{n-1} rỗi) then cấp phát tài nguyên cho P else if ($\{R_i \dots R_j\}$ được cấp phát cho Q và Q đang trong trạng thái chờ một số tài nguyên S khác) then thu hồi $\{R_i \dots R_j\}$ và cấp phát cho P else đưa P vào trạng thái chờ tài nguyên R_1, \dots, R_{n-1}

- Ngăn chặn “chờ vòng”

- + Một giải pháp ngăn chặn chờ vòng là đánh số thứ tự các tài nguyên và bắt buộc các tiến trình yêu cầu cấp phát tài nguyên theo số thứ tự tăng dần
- + Giả sử có các tài nguyên $\{R_1, \dots, R_n\}$. Ta gán cho mỗi tài nguyên một số nguyên dương duy nhất qua một ánh xạ 1-1 $f: R \rightarrow N$, với N là tập các số tự nhiên

- Giao thức ngăn chặn “Chờ vòng”:

- + Khi tiến trình P không chiếm giữ tài nguyên nào, nó có thể yêu cầu cấp phát nhiều thể hiện của một tài nguyên R_i bất kỳ
- + Sau đó P chỉ có thể yêu cầu các thể hiện của tài nguyên R_j nếu và chỉ nếu $f(R_j) > f(R_i)$. Một cách khác, nếu P muốn yêu cầu cấp phát tài nguyên R_j , nó đã giải phóng tất cả các tài nguyên R_i thỏa mãn $f(R_i) \geq f(R_j)$
- + Nếu P cần được cấp phát nhiều loại tài nguyên, P phải lần lượt yêu cầu các thể hiện của từng tài nguyên đó

- Ưu nhược điểm của ngăn chặn giải pháp bế tắc
 - + Ưu điểm: ngăn chặn bế tắc (deadlock prevention) là phương pháp tránh được bế tắc bằng cách làm cho điều kiện cần không được thỏa mãn
 - + Nhược điểm: Giảm khả năng tận dụng tài nguyên và giảm thông lượng của hệ thống và không mềm dẻo

1.2.2 Khái niệm dãy tiến trình an toàn

Cho 1 dãy tiến trình $P_1, P_2, P_3, \dots, P_n$ song hành. Dãy tiến trình gọi là an toàn (safe process) nếu với mọi tiến trình P_i , tài nguyên mà P_i cần có được thỏa mãn bởi các tài nguyên khả dụng của hệ thống và tài nguyên do các tiến trình P_i' đang giữ với điều kiện $i' < i$.

Hệ thống ở trạng thái an toàn tại một thời điểm nếu dãy tiến trình song hành tại thời điểm đó có thể được sắp xếp thành dãy an toàn.

Ví dụ:

Tiến trình	Max Need	Current Need
P_0	12	5
P_1	7	2
P_2	9	2

Đây là một dãy tiến trình an toàn.

Chương 2: Thuật toán kiểm tra tính an toàn hệ thống

2.1. Thuật toán chuyển sang trạng thái an toàn

Giả sử hệ có n tiến trình và m kiểu tài nguyên. Các cấu trúc dữ liệu sử dụng trong thuật toán được xây dựng như sau:

- Available: mảng $1 \times m$ thể hiện số tài nguyên có thể sử dụng của mỗi kiểu. Nếu $Available(j) = k$ suy ra có k tài nguyên kiểu r_j có thể sử dụng.
- Max: mảng $n \times m$ thể hiện số tài nguyên cực đại mà mỗi tiến trình yêu cầu. Nếu $Max(i, j) = k$ suy ra tiến trình P_i chỉ có thể yêu cầu cực đại k tài nguyên kiểu r_j cực đại là k .

- Allocation: mảng $n \times m$ thể hiện số tài nguyên mỗi kiểu hiện đã phân bổ cho các tiến trình. Nếu Allocation $(i, j) = k$ suy ra tiến trình P_i đang sử dụng k tài nguyên kiểu r_j .
- Need: mảng $n \times m$ thể hiện số tài nguyên còn cần của mỗi tiến trình. Need $(i, j) = k$; Tiến trình còn cần k tài nguyên của r_i .
- Need $(i, j) = \text{Max}(i, j) - \text{Allocation}(i, j)$
- Request: Mảng $n \times m$ thể hiện yêu cầu tài nguyên của các tiến trình
- Request $(i, j) = k$: tiến trình p_i yêu cầu k tài nguyên kiểu r_j .

Quy ước: khi viết request $_i$ thể hiện dòng thứ i của mảng (ứng với tiến trình p_i (các biến Need $_i$, Allocation $_i$, ...)).

Khi tiến trình p_i đưa ra một yêu cầu tài nguyên hệ thống tiến hành các bước như sau:

Step 1: If Request $_i \leq \text{Need}_i$ then goto Step 2

Else Error;

Step 2: If Request $_i \leq \text{Available}_i$ then goto Step 3

Else p_i đợi (Tài nguyên không thể phân bổ);

Step 3: Hệ thống dự định phân bổ tài nguyên

Available $_i = \text{Available}_i - \text{Request}_i$

Allocation $_i = \text{Allocation}_i + \text{Request}_i$

Need $_i = \text{Need}_i - \text{Request}_i$

Step 4: kiểm tra hệ có ở trạng thái an toàn hay không

Nếu có phân bổ tài nguyên theo dự tính

Nếu không tiến trình p_i phải đợi cùng với tài nguyên Request $_i$.

2.2. Thuật toán kiểm tra tính an toàn của hệ

Trong thuật toán sử dụng thêm hai cấu trúc dữ liệu sau:

- **Work**: mảng $1 \times m$ thể hiện số tài nguyên khả dụng của hệ thống và số tài nguyên do các tiến trình P_i đang sử dụng với điều kiện $i' < i$.

- **Finish**: mảng $1 \times n$ đánh dấu các tiến trình đã xét.

2.3. Thuật toán:

Step 1: Khởi tạo

$Work[j] = Available[j];$

for ($i=0, i \leq (n-1), i++$)

Nếu $Allocation[i, j] \neq 0$ với mọi j thì $Finish[i]=false$

Ngược lại $Finish[i]=true$

Step 2: Tìm i sao cho $Finish[i] = false$ và $Request(i, j) \leq Work[j]$

Nếu không tìm thấy, go to Step 3

Step 3: $Work := Work + Allocation(i)$

$Finish(i) := true$

Go to Step 1

Step 4: Nếu tồn tại i sao cho $Finish(i) = false$ thì hệ thống sẽ gặp bế tắc

2.4. Bài code của thuật toán

```
#include <iostream>
```

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
using namespace std;
```

```
void xuat(int n, int m, int a[10][10])
```

```
{
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            cout << a[i][j] << " ";
        }
        cout << "\n";
    }
}
```

```
void xuat1(int m, int a[])
```

```
{
    for(int i = 0; i < m; i++)
        cout << a[i] << " ";
}
```

```

void nhapmang2chieu(int n, int m, int a[10][10])
{
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
        {
            cin >> a[i][j];
        }
}

int main()
{
    //khai bao du lieu
    int available[10], maxx[10][10], allocation[10][10], need[10][10];
    int finish[10], work[10]; //fn =1 chua xet, 0 da xet
    int n, m, dem = 0, kt, antoan;
    //nhap du lieu
    cout << "\nNhap so tien trinh n: ";
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        finish[i] = 0;
        dem++;
    }
    cout << "\nNhap so kieu tai nguyen m: ";
    cin >> m;
    cout << "\nNhap so tai nguyen cua moi kieu (available): ";
    for(int j = 0; j < m; j++)
    {
        cin >> available[j];
    }
    cout << "\nNhap so tai nguyen moi kieu da phan bo (allocation):\n";
    nhapmang2chieu(n, m, allocation);
    cout << "\nNhap so tai nguyen cuc dai ma moi tien trinh yeu cau (max):\n";
    nhapmang2chieu(n, m, maxx);
    cout << "\n Allocation\n";
    xuat(n, m, allocation);
    cout << endl << "\n Max\n";
    xuat(n, m, maxx);
    cout << endl << "\n Need\n";
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
        {
            need[i][j] = maxx[i][j] - allocation[i][j];
        }
    xuat(n, m, need);
}

```

```

//Kiem tra an toan:
for(int i = 0; i < m; i++)
{
    work[i] = available[i];
}
while(dem != 0)
{
    antoan = 0;
    for (int i = 0; i < n; i++)
    {
        if(finish[i] == 0)
        {
            kt = 1;
            for (int j = 0; j < m; j++)
            {
                if (need[i][j] > work[j])
                {
                    kt = 0;
                    break;
                }
            }
            if(kt == 1)
            {
                cout << "\nProcess " << i << " ";
                finish[i] = 1;
                antoan = 1;
                dem--;
                for (int j = 0; j < m; j++)
                {
                    work[j] += allocation[i][j];
                }
                break;
            }
        }
    }
    if (antoan == 0)
    {
        cout << "\nTien trinh gay be tac."<<endl;
        break;
    }
    else
    {
        cout << "\nTien trinh an toan"<<endl;
    }
}

```

```

if (antoan == 0)
{
    cout << "\nHe gay be tac.\n";
    // break;
}
else
{
    cout << "\n ==> He an toan";
    cout << "\n";
}
getch();
}

```

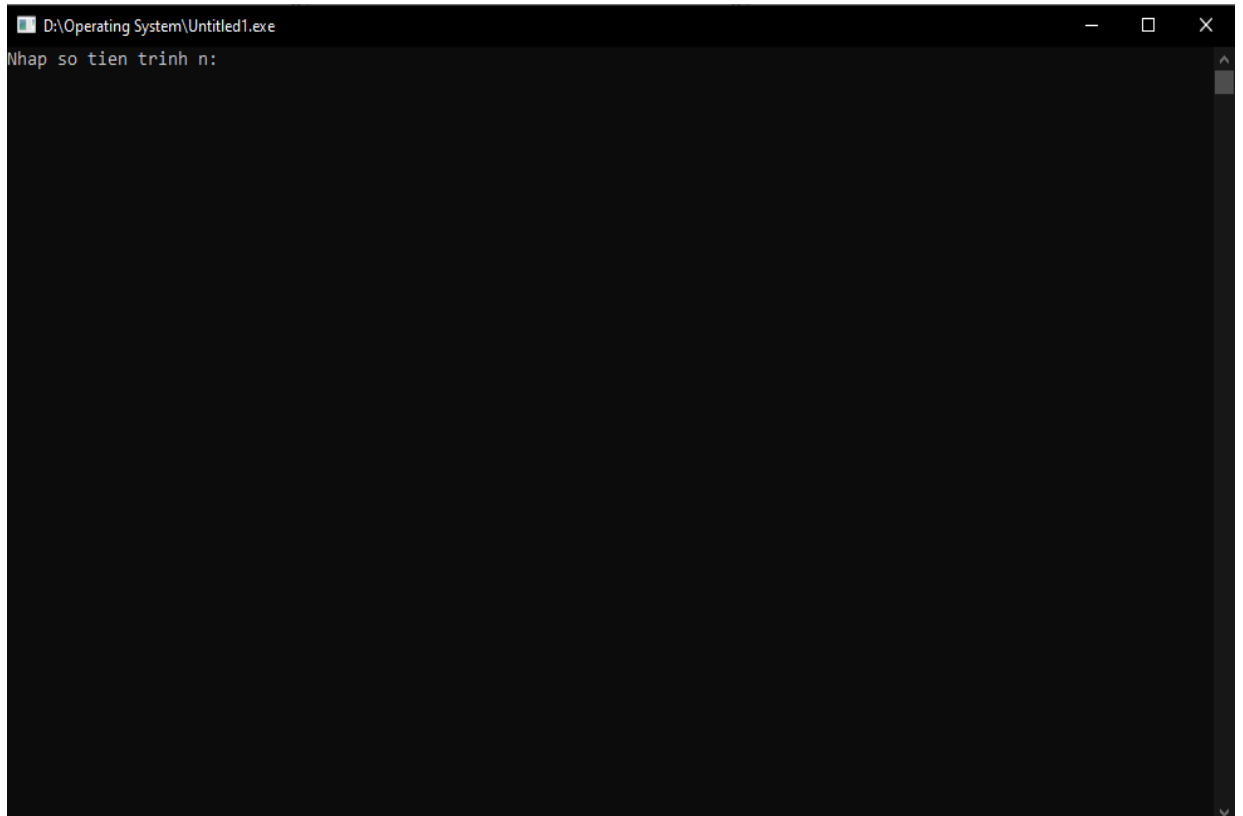
2.5. Ví dụ minh họa cho code

Cho dãy tiến trình và kiểu phân bố tài nguyên như sau

Tiến trình	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			

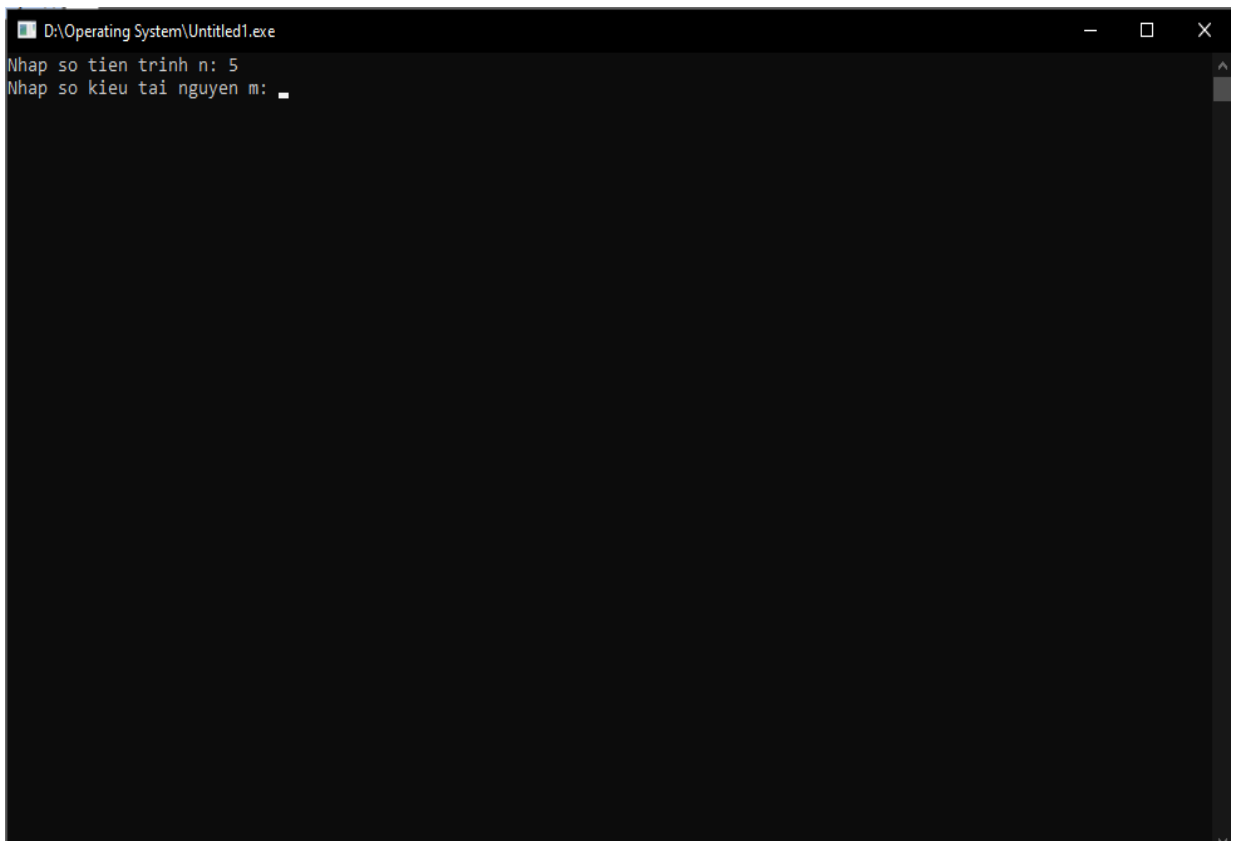
Kiểm tra xem hệ có ở trạng thái cân bằng hay không?

Đầu tiên, ta nhập số tiến trình



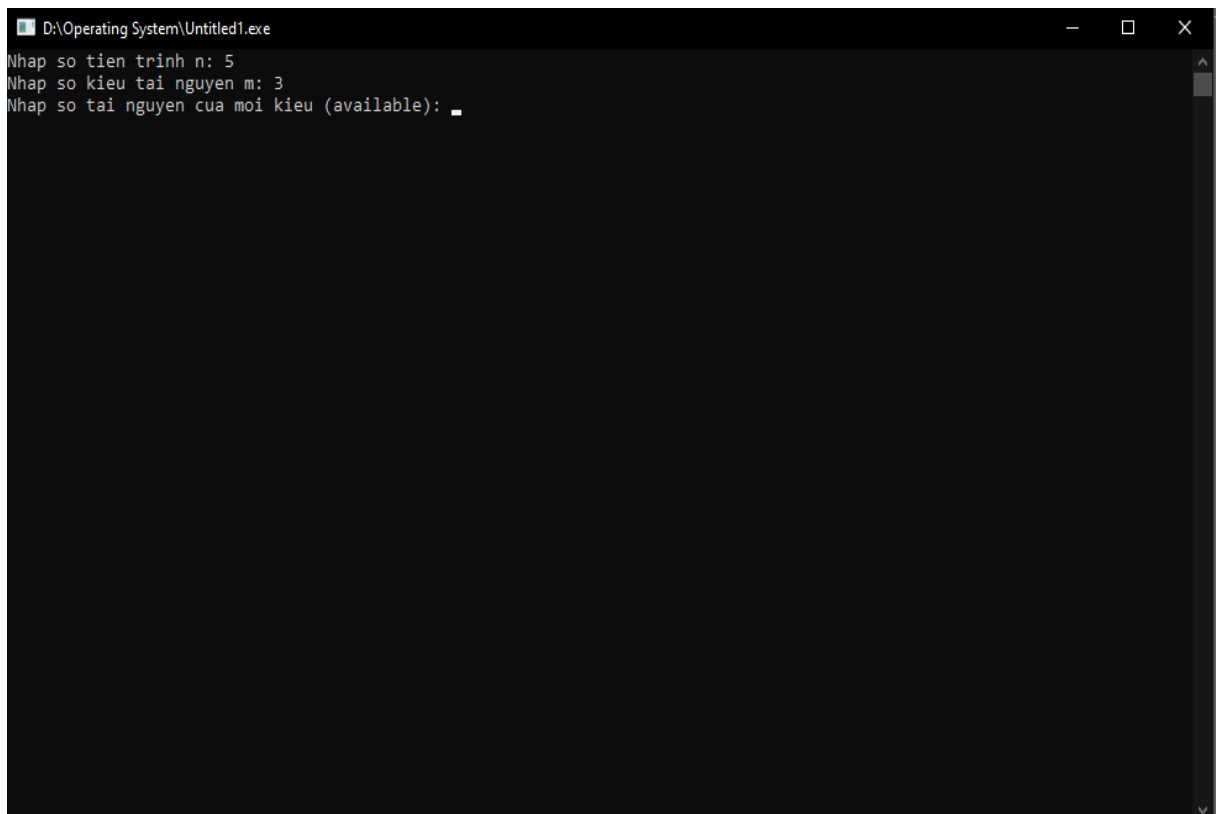
Hình 2.5.1 Nhập số tiến trình

Ta nhập số kiểu tài nguyên (đặt là m)



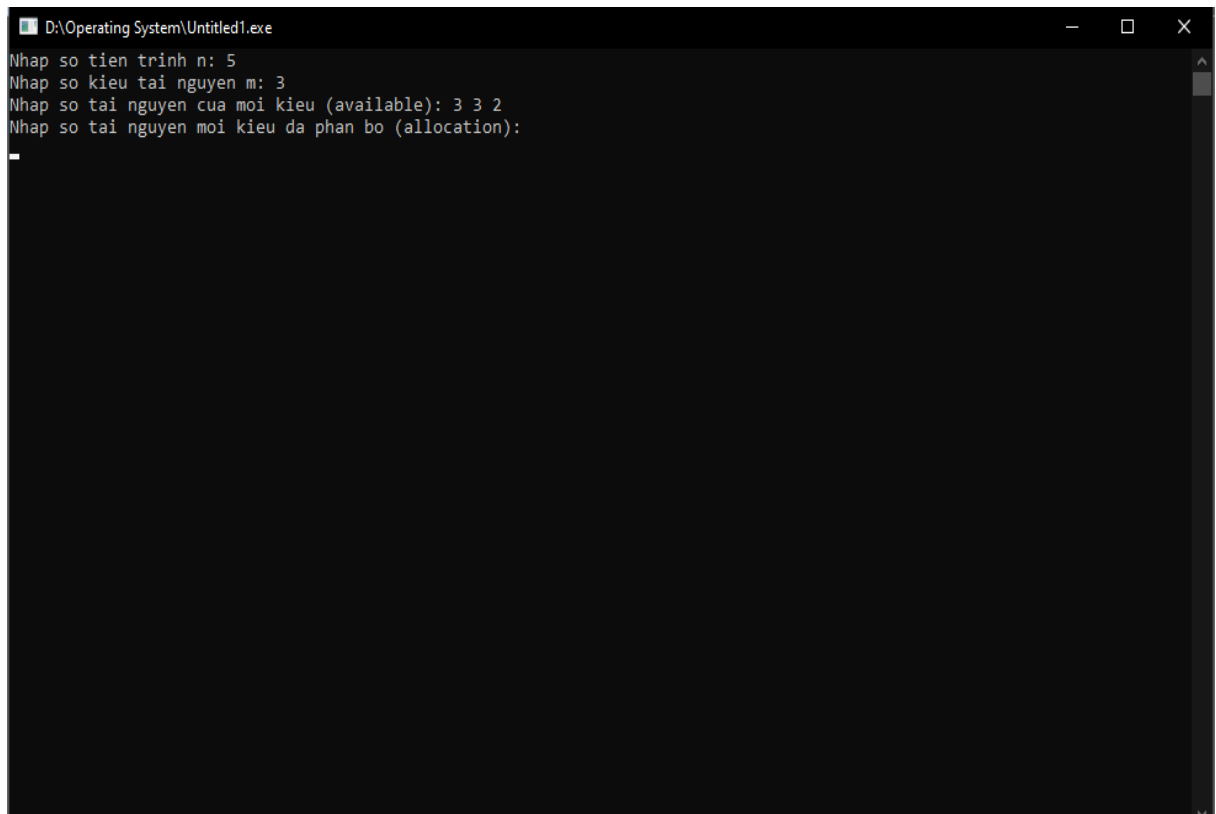
Hình 2.5.2 Nhập số tài kiểu tài nguyên

Ta nhập tài nguyên của mỗi kiểu (tức là bảng Available)




Hình 2.5.3 Nhập tài nguyên của mỗi kiểu (Available)

Ta nhập số tài nguyên mỗi kiểu đã phân bố (bảng Allocation):



Hình 2.5.4 Nhập tài nguyên của mỗi kiểu đã phân bổ (Allocation)

Ta nhập số tài nguyên cực đại mà mỗi tiến trình yêu cầu (bảng Max):



```
D:\Operating System\Untitled1.exe
Nhap so tien trinh n: 5
Nhap so kieu tai nguyen m: 3
Nhap so tai nguyen cua moi kieu (available): 3 3 2
Nhap so tai nguyen moi kieu da phan bo (allocation):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Nhap so tai nguyen cuc dai ma moi tien trinh yeu cau (max):
```

Hình 2.5.5 Nhập số tài nguyên cực đại (Max)

Ta nhập xong bảng Max thì được kết quả như sau:

```

D:\Operating System\Untitled1.exe
0 0 2
Nhap so tai nguyen cuc dai ma moi tien trinh yeu cau (max):
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Allocation
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Max
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Need
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

Process 1
Tien trinh an toan

Process 3
Tien trinh an toan

D:\OOP\Untitled1.exe
3 2 2
9 0 2
2 2 2
4 3 3

Need
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

Process 1
Tien trinh an toan

Process 3
Tien trinh an toan

Process 0
Tien trinh an toan

Process 2
Tien trinh an toan

Process 4
Tien trinh an toan

==> He an toan

```

Hình 2.5.6 Hiển thị kết quả trên màn hình Console

2.6. Ví dụ làm ở dạng lý thuyết

Cho 5 tiến trình và 3 kiểu tài nguyên ở trạng thái như sau:

Process	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

Hệ có ở trạng thái an toàn hay không?

Bài làm:

Để xét tính an toàn của hệ, ta cần tính ma trận Need theo công thức:

$$\text{Need} = \text{Max} - \text{Allocation}$$

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

Theo thuật toán kiểm tra tính an toàn của hệ ta có:

$$\text{Work} = \text{Available} = (3, 3, 2)$$

$$\text{Finish}[i] = \text{False} \text{ với } i = 0, 1, 2, 3, 4.$$

Xét:

$$\text{Need}[0] > \text{Work} \Rightarrow \text{finish}[0] = \text{false}$$

$$\text{Need}[2] > \text{Work} \Rightarrow \text{finish}[0] = \text{false}$$

$$\text{Need}[4] > \text{Work} \Rightarrow \text{finish}[0] = \text{false}$$

$$\text{Need}[0] = (7, 4, 3) > \text{Work} = (3, 3, 2)$$

$$\text{Need}[2] = (6, 0, 0) > \text{Work} = (3, 3, 2)$$

$$\text{Need}[4] = (4, 3, 1) > \text{Work} = (3, 3, 2)$$

$$\text{Need}[1] \leq \text{Work} \Rightarrow \text{Finish}[1] = \text{True} \text{ và } \text{Work} = \text{Work} + \text{Allocation}[1] = (3, 3, 2) + (2, 0, 0) = (5, 3, 2).$$

$$\text{Need}[3] \leq \text{Work} \Rightarrow \text{Finish}[3] = \text{True} \text{ và } \text{Work} = \text{Work} + \text{Allocation}[3] = (5, 3, 2) + (2, 1, 1) = (7, 4, 3).$$

$Need[0] \leq Work \Rightarrow Finish[0] = True$ và $Work = Work + Allocation[0] = (7,4,3) + (0,1,0) = (7,5,3)$.

$Need[2] \leq Work \Rightarrow Finish[2] = True$ và $Work = Work + Allocation[2] = (7,5,3) + (3,0,2) = (10,5,5)$.

$Need[4] \leq Work \Rightarrow Finish[4] = True$ và $Work = Work + Allocation[4] = (10,5,5) + (0,0,2) = (10,5,7)$.

Như vậy, $Finish[i] = true$ với $i = 1, 3, 0, 2, 4$ dẫn đến dãy tiến trình P1, P3, P0, P2, P4 là dãy an toàn suy ra hệ ở trạng thái an toàn.

KẾT LUẬN

Qua phần bài tập được trình bày trong tài liệu này, ta đã nắm được các nguyên lý trong cơ chế lập trình mô phỏng các phương pháp kiểm tra tính an toàn của hệ ở mức độ căn bản. Từ đó giúp chúng ta hiểu được một phần cách vận hành, hoạt động hệ thống của hệ điều hành. Tài liệu đã mang đến một phần kiến thức tuy nhỏ nhưng khá đầy đủ và căn bản để có thể giúp chúng em tự tin khi nghiên cứu sâu hơn trong vấn đề kiểm tra tính an toàn của hệ. Tài liệu này chủ yếu tìm hiểu về khái niệm, sơ lược, những kiến thức chung về bế tắc (deadlock) và các biện pháp ngăn chặn bế tắc (deadlock prevention), dãy tiến trình an toàn (safe process) và an toàn hệ thống, nghiên cứu về thuật toán kiểm tra tính an toàn của hệ, quản lý tiến trình như tạo một tiến trình, dừng một tiến trình hay việc giao tiếp, liên lạc giữa các tiến trình. Phần bài tập này chỉ mang tính chất nghiên cứu cơ bản nên không thực sự trình bày sâu nhưng chúng em đã cố gắng tóm lược lược các vấn đề chính để hiểu được cách hoạt động của các phương pháp kiểm tra tính an toàn của hệ ... Qua đó thấy được tầm quan trọng của việc kiểm tra, nghiên cứu tính an toàn của hệ.

TÀI LIỆU THAM KHẢO

- [1] Nguyễn Thanh Hải - *Giáo Trình Nguyên Lý Hệ Điều Hành* Đại học Công nghiệp Hà Nội, khoa công nghệ thông tin, Hà Nội 2016.
- [2] Nguyễn Kim Tuấn - *Giáo Trình Lý Thuyết Hệ Điều Hành* Đại học Huế, Trường Đại học khoa học, khoa công nghệ thông tin, Huế 06/2004.
- [3] Trần Hồ Thủy Tiên - *Giáo Trình Nguyên Lý Hệ Điều Hành* Đại học Đà Nẵng, trường Đại học Bách Khoa, Khoa Công Nghệ Thông Tin 01/04/2010.
- [4] Abraham Silberschatz, Galvin, Gagne, Operating System Concepts 8th edition. (tài liệu tham khảo từ nguồn nước ngoài).
- [5] <https://docs.microsoft.com/en-us/windowshardware/drivers/devtest/deadlock-detection>.