

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

-----🙞🙜🕮🙞🙜-----



**BÁO CÁO MÔN HỌC: NGUYÊN LÝ HỆ ĐIỀU HÀNH**

**ĐỀ TÀI: LẬP TRÌNH MÔ PHỎNG PHƯƠNG PHÁP PHÂN BỔ TÀI NGUYÊN ĐỂ GIẢI QUYẾT BẾ TẮC**

Giảng viên: ***Ths Nguyễn Bá Nghiễn***

Nhóm: 13

Thành viên nhóm: *Đặng Thìn Hiếu*

*Nguyễn Trọng Tú Tâm*

*Nguyễn Quang Minh*

*Hoàng Đức Nam*

Lớp: 20212IT6025004 – K15

Hà Nội, 2022

**MỤC LỤC**

[**Lời mở đầu** 3](#_heading=h.gjdgxs)

[**Giới thiệu chung** 4](#_heading=h.30j0zll)

[**Chương 1: Hiện tượng bế tắc** 4](#_heading=h.1fob9te)

[**1.1**](#_heading=h.lnxbz9) **Các khái niệm cơ bản** 4

[**1.1.1**](#_heading=h.3znysh7) **Hiện tượng bế tắc** 4

[**1.1.2**](#_heading=h.2et92p0) **Điều kiện xảy ra bế tắc** 4

[**1.2**](#_heading=h.tyjcwt) **Các biện pháp phòng tránh bế tắc** 5

[**1.2.1**](#_heading=h.3dy6vkm) **Ngăn chặn bế tắc** 5

[**1.2.2**](#_heading=h.1t3h5sf) **Dự báo và tránh bế tắc** 7

[**1.2.3**](#_heading=h.4d34og8) **Xử lý bế tắc** 7

[**Chương 2 Mô phỏng phương pháp phân bổ tài nguyên để giải quyết bế tắc** 9](#_heading=h.2s8eyo1)

[**2.1**](#_heading=h.17dp8vu) **Thuật toán phân bổ tài nguyên để giải quyết bế tắc(Thuật toán Banker)** 9

[**2.2**](#_heading=h.3rdcrjn) **Source code** 10

[**2.3**](#_heading=h.35nkun2) **Hình ảnh minh họa cho thuật toán** 13

[**2.4**](#_heading=h.1ksv4uv) **Giải thích thuật toán bằng ví dụ** 16

[**Tài liệu tham khảo** 18](#_heading=h.26in1rg)

# **Lời mở đầu**

Lời đầu tiên, chúng em muốn gửi những lời cảm ơn và biết ơn chân thành nhất của mình tới tất cả những người đã hỗ trợ, giúp đỡ chúng em về kiến thức và tinh thần trong quá trình thực hiện đồ án.

Có thể nói rằng, trong những năm gần đây khoa học công nghệ trí tuệ loài người không ngừng nâng lên tầm cao mới, còn thành tựu khoa học thì được phát minh hàng ngày hàng giờ nhằm phục vụ cho những nghiên cứu tương lai. Không nằm ngoài nhịp vận động của thời đại, Công Nghệ Thông Tin cũng phát triển như vũ bão với hàng loạt những tiện ích phục vụ ngày càng hiệu quả cho cuộc sống. Máy tính điện tử hiện nay không còn là món hàng xa xỉ mà ngày càng trở thành một công cụ làm việc và giải trí đắc lực của của con người. Mặt khác, đất nước ta đang trên đường công nghiệp hóa, hiện đại hóa thì Tin học hóa các ngành là vấn đề cấp bách. Bất kỳ ngành nghề nào cũng đòi hỏi những yêu cầu nhất định, tin học không chỉ được áp dụng đại trà mà quan trọng nhất là phải đáp ứng được đúng các yêu cầu vốn rất đa dạng của người dùng. Để thực hiện được điều đó thì việc phân tích hệ điều hành để chọn ra hương đi phù hợp với thực tiễn là rất quan trọng và luôn là thách thức đối với những ai làm phần mềm.

Ngày nay, trong thời đại lĩnh vực Công Nghệ Thông Tin bùng nổ trên toàn cầu thì các quốc gia trên thế giới đều cố gắng áp dụng tin học vào đề hiện đại hoá quy trình sản xuất kinh doanh của mình nhằm đạt hiệu quả cao nhất. *Nguyên lý hệ điều hành* là một trong những môn học được quan tâm nhiều nhất trong khoa học về máy tính.

Chúng em xin chân thành cảm ơn!

# **Giới thiệu chung**

Hệ điều hành là một tập hợp các phần mềm hệ thống điều khiển mọi hoạt động của máy tính và tạo môi trường giao diện giữa người dùng và máy tính. Vì vậy hệ điều hành rất quen thuộc với người sử dụng; tất cả mọi người dùng khi làm việc với máy tính cần phải biết thao tác (dù là những thao tác đơn giản) với một hệ điều hành cụ thể.

Hiểu biết, nắm vững nguyên lí hoạt động của hệ điều hành để từ đó có thể khai thác các hệ điều hành hiệu quả là một nhu cầu thực tế không thể thiếu đối với các cán bộ phụ trách kĩ thuật và quản lí các hệ thống máy tính.

**Chương 1: Hiện tượng bế tắc**

## **Các khái niệm cơ bản**

### **Hiện tượng bế tắc**

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.

### **Điều kiện xảy ra bế tắc**

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 rỗi 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 rỗi khi tiến trình “tự nguyện” giải phóng tài nguyên sau khi đã sử dụng xong.

- Chờ vòng (circular wait)

* Một tập tiến trình {P0, P1, ..., Pn} có xuất hiện điều kiện “chờ vòng” nếu P0 chờ một tài nguyên do P1 chiếm giữ, P1 chờ một tài nguyên khác do P2 chiếm giữ, ..., Pn-1 chờ tài nguyên do Pn chiếm giữ và Pn chờ tài nguyên do P0 chiếm giữ

## **Ngăn chặn bế tắc**

### **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 R1, ..., Rn-1) yêu cầu cấp phát tài nguyên Rn nhưng không được cấp phát ngay (có nghĩa là P phải chờ) thì tất cả các tài nguyên R1, ..., Rn-1 phải được “thu hồi”
* Nói cách khác, R1, ..., Rn-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 R1, ..., Rn-1 if (R1, ..., Rn-1 rỗi) then cấp phát tài nguyên cho P else if ({Ri ...Rj} được cấp phát cho Q và Q đang trong trạng thái chờ một số tài nguyên S khác) then thu hồi {Ri ...Rj} và cấp phát cho P else đưa P vào trạng thái chờ tài nguyên R1, ..., Rn-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 {R1, ..., Rn}. Ta gán cho mỗi tài nguyên một số nguyên dương duy nhất qua một ánh xạ 1-1 f: R → 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 Ri bất kỳ
* Sau đó P chỉ có thể yêu cầu các thể hiện của tài nguyên Rj nếu và chỉ nếu f(Rj) > f(Ri). Một cách khác, nếu P muốn yêu cầu cấp phát tài nguyên Rj, nó đã giải phóng tất cả các tài nguyên Ri thỏa mãn f(Ri) ≥ f(Rj)
* 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 giải pháp ngăn chặn 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

### **Dự báo và tránh bế tắc**

Để ngăn chặn thì làm cho hệ thống không xảy ra đồng thời 4 điều kiện bế tắc:

* Loại bỏ tài nguyên găng: mô phỏng tài nguyên găng bằng các tài nguyên có thể dùng chung được
* Loại bỏ yếu tố giữ và đợi: thực hiên phân bổ tài nguyên, nếu tiến trình phải đợi thì mọi tài nguyên nó đang giữ tạm thời giải phóng
* Xây dựng hệ thống ngắt tài nguyên
* Loại bỏ yếu tố vòng đợi: vòng đợi có thể loại bỏ bằng cách sắp thứ tự các tài nguyên

### **Xử lý bế tắc**

Khi đã phát hiện được bế tắc, HĐH có thể áp dụng các phương pháp giải quyết:

* Thông báo cho operator để biết tự xử lý
* Đình chỉ hoạt động các tiến trình có liên quan, thu hồi tất cả các tài nguyên bị kết thúc và sử dụng 2 phương pháp 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

# **Chương 2: Mô phỏng phương pháp phân bổ tài nguyên để giải quyết bế tắc**

1. **Thuật toán phân bổ tài nguyên để giải quyết bế tắc(Thuật toán Banker)**

Giải thuật đồ thị cấp phát tài nguyên không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên. Giải thuật tránh deadlock mà chúng ta mô tả tiếp theo có thể áp dụng tới một hệ thống nhưng ít hiệu quả hơn cơ chế đồ thị cấp phát tài nguyên. Giải thuật này thường được gọi là giải thuật của Banker. Tên được chọn vì giải thuật này có thể được dùng trong hệ thống ngân hàng để đảm bảo ngân hàng không bao giờ cấp phát tiền mặt đang có của nó khi nó không thể thoả mãn các yêu cầu của tất cả khách hàng.

Khi một quá trình mới đưa vào hệ thống, nó phải khai báo số tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này có thể không vượt quá tổng số tài nguyên trong hệ thống. Khi một người dùng yêu cầu tập hợp các tài nguyên, hệ thống phải xác định việc cấp phát của các tài nguyên này sẽ để lại hệ thống ở trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ là an toàn, tài nguyên sẽ được cấp, ngược lại quá trình phải chờ cho tới khi một vài quá trình giải phóng đủ tài nguyên.

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 1x m thể hiện số tài nguyên có thể sử dụng của mỗi kiểu. Nếu Available (j) = k suy ra có k tài nguyên kiểu rj có thể sử dụng.

- Max: mảng n x m thể hiện số tài nguyên cực đại mà mỗi tiến trình yêu cầu. Nếu Max (i, j) = k suy ra tiến trình Pi chỉ có thể yêu cầu cực đại k tài nguyên kiểu rjcực đại là k.

- Allocation: mảng n x 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 Pi đang sử dụng k tài nguyên kiểu rj.

- Need: mảng n x m thể hiện số tài nguyên còn cần của mỗi tiến trình. Need (i, j) = k; Tiến trình còn cần k tài nguyên của ri.

- Need (i, j) = Max (i, j) – Allocation (i, j)

- Request: Mảng n x m thể hiện yêu cầu tài nguyên của các tiến trình

- Request (i, j) = k: tiến trình pi yều cầu k tài nguyên kiểu rj.

Quy ước: khi viết request i thể hiện dòng thứ i của mảng (ứng với tiến trình pi (các biến Needi, Allocationi,…)).

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

Step 1: If Request i<=Need i then goto Step2

Else Error;

Step 2: If Request i<=Available i then goto Step 3

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

Step 3: Hệ thống dự định phân bổ tài nguyên

Available i = Available i – Request i

Allocation i = Allocation i – Request i

Need i = Need i – Request i

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

Nếu có phân bổ tài nguyên theo dự tính

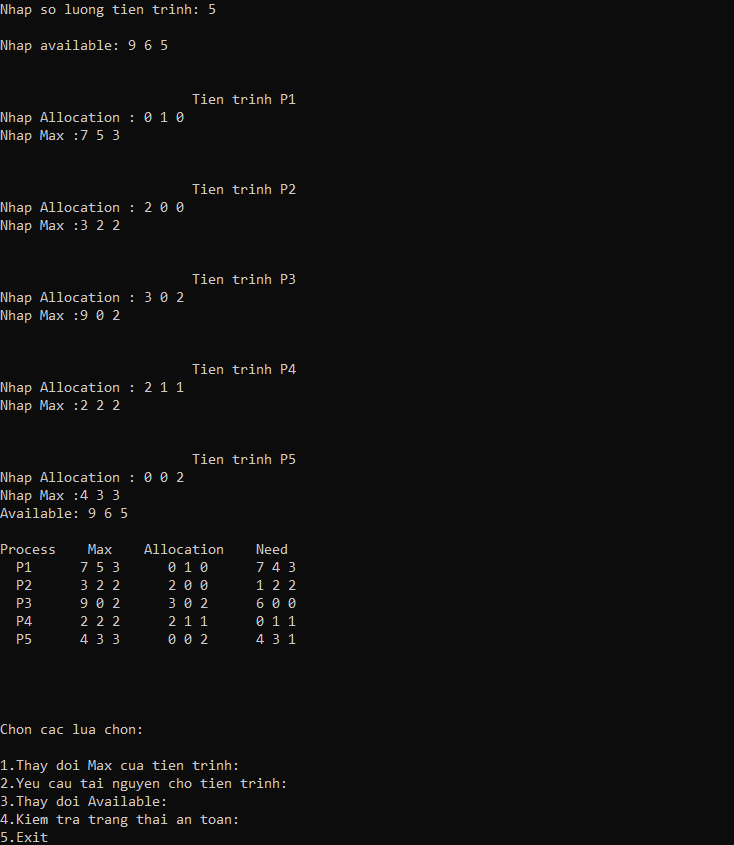
Nếu không tiến trình pi phải đợi cùng với tài nguyên Requesti.

1. **Source code**

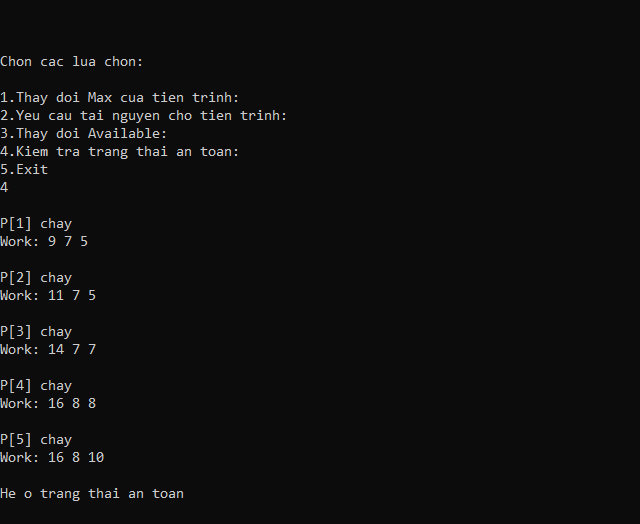
| #include<bits/stdc++.h>  using namespace std;  void Calculate();  void initial();  int allocation[10][3],need[10][3],Max[10][3],request[10][3];  int p,available[3];  bool executed[10],come;  int main (){    int keepon = 1;  cout<<"Nhap so luong tien trinh: ";  cin>>p;  cout<<"\n";  cout<<"Nhap available: ";  cin>>available[0]>>available[1]>>available[2];    for (int i = 0; i < p; i++)  {  cout<<"\n\n\t\t\tTien trinh P"<<i+1<<"\n";  cout<<"Nhap Allocation : ";  cin>>allocation[i][0]>>allocation[i][1]>>allocation[i][2];  cout<<"Nhap Max :";  cin>>Max[i][0]>>Max[i][1]>>Max[i][2];  need[i][0]=Max[i][0]-allocation[i][0];  need[i][1]=Max[i][1]-allocation[i][1];  need[i][2]=Max[i][2]-allocation[i][2];  }    cout<<"Available: "<<available[0]<<" "<<available[1]<<" "<<available[2]<<"\n\n";  cout<<"Process Max Allocation Need\n";    for (int i = 0; i < p; i++)  {  cout<<" P"<<i+1<<" ";  cout<<" "<<Max[i][0]<<" "<<Max[i][1]<<" "<<Max[i][2]<<" ";  cout<<" "<<allocation[i][0]<<" "<<allocation[i][1]<<" "<<allocation[i][2]<<" ";  cout<<" "<<need[i][0]<<" "<<need[i][1]<<" "<<need[i][2];  cout<<"\n";  }  cout<<"\n\n";    while(keepon){  int val,pro;  cout<<"\n\nChon cac lua chon:\n\n";  cout<<"1.Thay doi Max cua tien trinh: \n";  cout<<"2.Yeu cau tai nguyen cho tien trinh: \n";  cout<<"3.Thay doi Available: \n";  cout<<"4.Kiem tra trang thai an toan: \n";  cout<<"5.Exit\n";  cin>>val;  switch(val)  {  case 1:  cout<<"\n\nNhap tien trinh thu: ";  cin>>pro;  cout<<"\nNhap gia tri Max moi: ";  cin>>Max[pro-1][0]>>Max[pro-1][1]>>Max[pro-1][2];  break;  case 2:  cout<<"\n\nNhap tien trinh thu: ";  cin>>pro;  cout<<"\nNhap gia tri Request: ";  cin>>request[pro-1][0]>>request[pro-1][1]>>request[pro-1][2];    if(request[pro-1][0]<=need[pro-1][0] && request[pro-1][1]<=need[pro-1][1] && request[pro-1][2]<=need[pro-1][2]) {  if(request[pro-1][0]<=available[0] && request[pro-1][1]<=available[1] && request[pro-1][2]<=available[2]) {  available[0]-=request[pro-1][0];  available[1]-=request[pro-1][1];  available[2]-=request[pro-1][2];  allocation[pro-1][0]+=request[pro-1][0];  allocation[pro-1][1]+=request[pro-1][1];  allocation[pro-1][2]+=request[pro-1][2];  need[pro-1][0]-=request[pro-1][0];  need[pro-1][1]-=request[pro-1][1];  need[pro-1][2]-=request[pro-1][2];  cout<< "Co the thuc hien request \n";  }  else cout << "Process "<< pro << " wait\n";  }  else cout << "Khong thuc hien duoc request\n";  break;  case 3:  cout<<"\nNhap Available: ";  cin>>available[0]>>available[1]>>available[2];  break;  case 4:  Calculate();  break;  case 5:  exit(0);  }    }  return 0;  }  void Calculate(){  IMP();  int i,j;  int count = 0;    for (i = 0; i < p; i++)  {  for (j = 0; j < p; j++)  {  while(executed[j] && j<p-1){  j++;  }  if (need[j][0]<=available[0]&&need[j][1]<=available[1]&&need[j][2]<=available[2])  {  if (!executed[j])  {  executed[j]=true;  available[0]+=allocation[j][0];  available[1]+=allocation[j][1];  available[2]+=allocation[j][2];  cout<<"\nP["<<j+1<<"] chay";  cout<<"\nWork: "<<available[0]<<" "<<available[1]<<" "<<available[2]<<"\n";  come=true;    break;  }  }  }  if (!come)  {  cout<<"\n\t\t\tDead lock\n\n" <<p << "--" << count;  break;  }else{  come=false;  }  }  for (i = 0; i < p; i++){  if (executed[i]) count++;  }  if (count == p) cout<<"\nHe o trang thai an toan\n";  else cout<<"\nHe khong o trang thai an toan\n";  }  void initial(){  come=false;  for (int i = 0; i < 10; ++i)  {  executed[i]=false;  }  } |
| --- |

1. **Hình ảnh minh họa cho thuật toán**

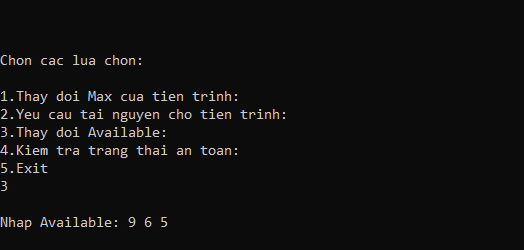
Nhập số tiến trình và số tài nguyên đã phân bổ cho từng tiến trình:



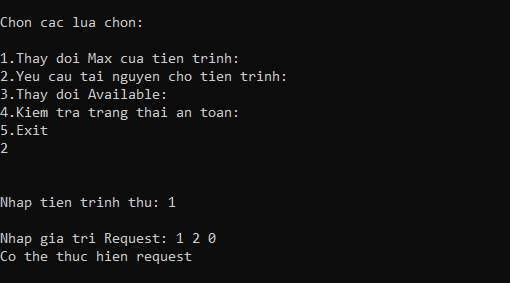
Hình 2.3.1. Giao diện chính sau khi chạy chương trình



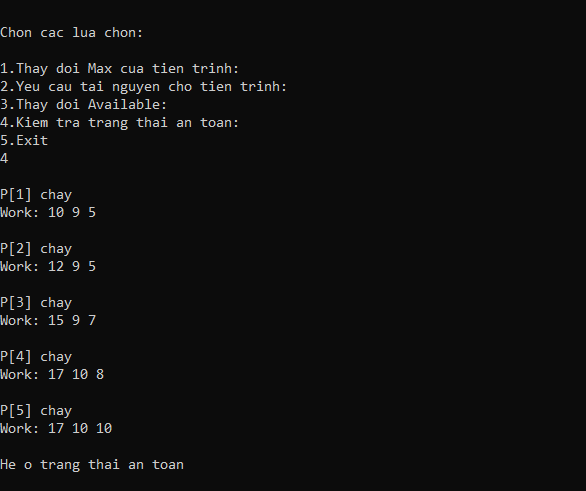
Hình 2.3.2. Kiểm tra trạng thái của hệ:



Hình 2.3.3. Nhập lại available:



Hình 2.3.4. Yêu cầu tài nguyên cho tiến trình:



Hình 2.3.5. Kiểm tra trạng thái của hệ sau khi yêu cầu:

1. **Giải thích thuật toán bằng ví dụ**

| **Process** | **Allocation** | | | **Need** | | | **Available** | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | A | B | C | A | B | C |
| **P0** | 3 | 0 | 2 | 4 | 0 | 1 | 2 | 1 | 0 |
| **P1** | 2 | 1 | 0 | 7 | 1 | 2 |  |  |  |
| **P2** | 0 | 1 | 1 | 2 | 4 | 2 |  |  |  |
| **P3** | 0 | 0 | 2 | 4 | 3 | 0 |  |  |  |
| **P4** | 3 | 2 | 2 | 0 | 0 | 0 |  |  |  |

Hình 2.4.1 Bảng trạng thái của hệ

Yêu cầu tài nguyên(1, 2, 2) cho tiến trình p4

Bước 1: Request[4] <= Need[4] => thỏa mãn

Bước 2: Request[4] <= Available[4] => thỏa mãn

Bước 3: Hệ thống dự định phân bổ như sau:

Available = Available – Request[4]

Allocation[4] = Allocation[4] + Request[4] = (3, 2, 2)

Need[4] = Need[4] - + Request[4]

Và hệ có trạng thái như hình 2.4.1

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

Work: Available = (2, 1, 0)

Finish[i] = false với i= 0, 1, 2, 3, 4

Xét:

Need[0] >= Work => Finish[0] = false

Need[1] >= Work => Finish[1] = false

Need[2] >= Work => Finish[2] = false

Need[3] >= Work => Finish[3] = false

Need[4] <= Work => Finish[4] = true

Và work = work + allocation[4] = (5, 3, 2)

Need[0] <= Work => Finish[0] = true

Và work = work + allocation[0] = (8, 3, 4)

Need[1] <= Work => Finish[1] = true

Và work = work + allocation[1] = (10, 4, 4)

Need[2] <= Work => Finish[2] = true

Và work = work + allocation[2] = (10, 5, 5)

Need[3] <= Work => Finish[3] = true

Và work = work + allocation[3] = (12, 5, 5)

Như vậy, Finish[i] = true với i=4, 0, 1, 2, 3 dẫn đến dãy tiến trình p4, p0, p1, p2, p3 ở trạng thái an toàn suy ra hệ ở trạng thái an toàn. Do đó có thể phân bổ tài nguyên cho p4.

**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 nghê 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.