TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THỰC PHẨM TP.HCM KHOA CÔNG NGHỆ THÔNG TIN

-----o0o-----

TÊN NHÓM: 10

TÊN ĐỀ TÀI: TÌM HIỂU GIẢI THUẬT QUẢN LÝ TẮC NGHẼN VÀ VIẾT CHƯƠNG TRÌNH MINH HỌA

BÀI TẬP CUỐI KÌ MÔN: HỆ ĐIỀU HÀNH

TP.HCM, NĂM 2021

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THỰC PHẨM TP.HCM KHOA CÔNG NGHỆ THÔNG TIN

-----o0o-----

TÊN NHÓM: 10

TÊN ĐỀ TÀI: TÌM HIỂU GIẢI THUẬT QUẢN LÝ TẮC NGHẼN VÀ VIẾT CHƯƠNG TRÌNH MINH HỌA

BÀI TẬP CUỐI KÌ MÔN: HỆ ĐIỀU HÀNH

GVHD: TS. Trần Thị Bích Vân

Lớp danh nghĩa: 11DHTH1

NHÓM THỰC HIỆN:

1. **Đỗ Huệ Mẫn – 2001202148**
2. **Lê Trần Tiến – 2001207261**
3. **Trần Long Hồ - 2001200064**
4. **Bùi Trần Minh Kha– 2001207104**

TP.HCM, NĂM 2021

BẢNG ĐÁNH GIÁ KẾT QUẢ THỰC HIỆN CÔNG VIỆC NHÓM

STT	Họ và tên	Công việc đảm nhận	Cá nhân tự đánh giá kết quả	Nhóm đánh giá kết quả	GV đánh giá
1	Bùi Trần Minh Kha	Chương 1: Tổng quan hệ điều hành + Chương 2: Tìm hiểu tài nguyên (phần 2.1) + Tìm hiểu khái niệm tắc nghẽn (phần 2.2) + viết chương trình giải thuật phát hiện tắc nghẽn	Hoàn thành tốt, đúng hạn	100%	
2	Lê Trần Tiến	Chương 2: Tìm hiểu tắc nghẽn trong hệ điều hành (phần 2.3 + 2.4 + 2.5) + viết chương trình giải thuật Banker	Hoàn thành tốt, đúng hạn	100%	
3	Trần Long Hồ	Chương 3: Giải thuật Banker + viết chương trình giải thuật phát hiện tắc nghẽn	Hoàn thành tốt, đúng hạn	100%	
4	Đỗ Huệ Mẫn	Chương 4: Giải thuật phát hiện tắc nghẽn + viết báo cáo word + viết chương trình giải thuật Banker	Hoàn thành tốt, đúng hạn	100%	

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin được trân trọng gửi lời cảm ơn đến giáo viên hướng dẫn của chúng em là cô Trần Thị Bích Vân. Chúng em xin được bày tỏ lòng biết ơn chân thành nhất đối với cô giáo hướng dẫn, cô đã tận tình hướng dẫn chúng em trong quá trình học tập, giảng dạy chi tiết để chúng em có đủ kiến thức và vận dụng chúng vào bài tập này.

Chúng em cảm ơn trường đại học Công Nghiệp Thực Phẩm TP.Hồ Chí Minh đã tạo điều kiện và môi trường học tập tốt nhất để chúng em có thể học tập, vui chơi và nghiên cứu tại trường

Chúng em xin chân thành gửi đến lời cảm ơn đến quý thầy cô khoa công nghệ thông tin trường đại học Công Nghiệp Thực Phẩm TP.Hồ Chí Minh, thầy cô đã luôn tạo điều kiện thuận lợi nhất trong suốt thời gian chúng em học tập và nghiên cứu tại trường. Rất mong nhận được sự nhận xét, ý kiến, đóng góp, phê bình từ phía thầy, cô để bài tập này được hoàn thiện hơn.

Cuối cùng, xin gửi lời cảm ơn đến gia đình, bạn bè, tập thể lớp 11DHTH1, những người luôn sẵn sàng giúp đỡ và chia sẻ trong học tập và cuộc sống. Xin chúc những điều tốt đẹp nhất sẽ luôn đồng hành cùng mọi người. Lời cuối cùng, chúng em xin kính chúc thầy, cô nhiều sức khỏe, thành công và hạnh phúc.

Xin trân trọng cảm ơn!

MỤC LỤC

PHẦN MỞ ĐẦU

1. Lí do chọn đề tài

Hệ điều hành thuộc chương trình hệ thống là một chương trình quan trọng đối với máy tính và người sử dụng. Hệ điều hành cung cấp tất cả tài nguyên cho máy tính và cung cấp môi trường để các chương trình ứng dụng do người sử dụng viết ra có thể chạy được trên máy tính.

Trong môi trường đa chương, có nhiều tiến trình có thể cạnh tranh lẫn nhau bởi một số giới hạn về tài nguyên. Một tiến trình yêu cầu tài nguyên hệ thống, nếu tài nguyên đó đang được sử dụng không có sẵn thì tiến trình đó rơi vào trạng thái chờ đợi. Nếu không được xử lý thì các tiến trình sẽ chờ đợi nhau và không có tiến trình nào thực hiện tiếp. Vấn đề tắc nghẽn trở nên phổ biến không còn xa lạ với chúng ta, hầu hết các hệ điều hành không cung cấp phương tiện ngăn chặn tắc nghẽn, vì vậy chúng em sẽ tìm hiểu về đề tài: "Tìm hiểu giải thuật quản lý tắc nghẽn và viết chương trình minh họa" Trong phần này, chúng ta sẽ xem xét các tắc nghẽn kỹ hơn, xem chúng phát sinh như thế nào và nghiên cứu một số cách ngăn ngừa hoặc phòng tránh tắc nghẽn.

2. Mục tiêu

- Tìm hiểu về tắc nghẽn trong hệ điều hành
- Hiểu các quản lý tắc nghẽn trong hệ điều hành
- Tìm hiểu giải thuật Banker tránh tắc nghẽn và phát hiện tắc nghẽn
- Viết chương trình Banker và phát hiện tắc nghẽn

3. Phạm vi nghiên cứu

- Nghiên cứu lý thuyết môn học "Hệ điều hành"
- Nghiên cứu trọng tâm giải thuật Banker và giải thuật phát hiện tắc nghẽn
- Cài đặt chương trình Banker và phát hiện tắc nghẽn

4. Đối tượng nghiên cứu

- Tắc nghẽn trong hệ điều hành
- Giải thuật Banker
- Giải thuật phát hiện tắc nghẽn

5. Bố cục và nội dung chi tiết

Ngoài phần mở đầu và kết luận, nội dung của bài tập nhóm được trình bày trong 4 chương có cấu trúc như sau:

Chương 1: Tổng quan hệ điều hành

Chương 2: Tắc nghẽn trong hệ điều hành

Chương 3: Giải thuật tránh tắc nghẽn Banker

Chương 4: Giải thuật phát hiện tắc nghẽn

Chương 5: Chương trình giải thuật Banker và phát hiện tắc nghẽn

CHƯƠNG 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

1.1 Hệ điều hành là gì?

1.1.1 Khái niệm

- Hệ điều hành là một chương trình hoạt động trung gian giữa người sử dụng và phần cứng máy tính

- Hệ điều hành là một phần quan trọng của hầu hết các hệ thống máy tính. Một hệ thống máy tính thường được chia làm bốn phần chính : phần cứng, hệ điều hành, các chương trình ứng dụng và người sử dụng

1.1.2 Các thành phần của hệ điều hành

- Hệ thống quản lý tiến trình
- Hệ thống quản lý bộ nhớ
- Hệ thống quản lý nhập xuất
- Hệ thống quản lý tập tin
- Hệ thống bảo vệ
- Hệ thống dịch lệnh
- Quản lý mạng
- Ba thành phần quan trọng nhất của hệ điều hành đó là:

+ Kernel: Cung cấp những điều khiển cơ bản trên cấu hình phần cứng máy tính, từ đó đảm nhiệm các vai trò gồm: đọc - ghi dữ liệu, xử lý các lệnh, xác định dữ liệu được nhận và gửi bởi các thiết bị khác, đồng thời diễn giải dữ liệu nhận từ mạng.

+ User Interface (Giao diện người dùng): Đảm bảo quá trình tương tác giữa người dùng với máy tính thông qua Desktop, Graphical Icons hay Command Line.

+ Application Programming Interfaces (Giao diện lập trình ứng dụng): cho phép các ứng dụng phát triển sử dụng Modular Code

1.2 Lịch sử phát triển hệ điều hành

- Hệ điều hành đã và đang phát triển qua 4 thế hệ:

+ Thế hệ 1 (1945 – 1955): Vào khoảng những năm 1940, Howard Aiken ở Havard và John von Neumann ở Princeton, đã thành công trong việc xây dựng

máy tính dùng ống chân không. Những máy này rất lớn với hơn 10000 ống chân không nhưng chậm hơn nhiều so với máy tính rẻ nhất ngày nay.Mỗi máy được một nhóm thực hiện tất cả từ thiết kế, cho đến xây dựng lập trình và thao tác đến quản lý. Các hệ thống điện tử thời này được lập trình trên hàng công tắc cơ học hoặc bằng dây nhảy trên bảng cắm. Đây là những hệ thống có mục đích đặc biệt

+ Thế hệ 2 (1955 – 1965): Hệ thống xử lý theo lô ra đời, nó lưu các yêu cầu cần phải thực hiện lên trên băng từ, và hệ thống sẽ tiến hành đọc và thi hành lần lượt. Sau đó, nó sẽ ghi kết quả lên trên bằng từ xuất và cuối cùng người sử dụng sẽ đem băng từ xuất đi in

+ Thế hệ 3 (1965 – 1980): Trong giai đoạn này với sự ra đời của máy IBM 360 là máy đầu tiên sử dụng mạch tích hợp (IC), công suất máy tính tăng 1 cách đáng kể và từ đó kích thước và giá cả giảm đáng kể và càng ngày dễ phổ biến hơn. Hệ điều hành với các chức năng như ta thường thấy ngày nay ra đời ở giai đoạng này nhằm điều phối, kiểm soát hoạt động và giải quyết các vấn đề tranh chấp thiết bị.

+ Thế hệ 4 (sau năm 1980): Đây là giai đoạn đánh dấu sự ra đời của máy tính cá nhân và là giai đoạn phát triển bùng nổ của máy tính. Những hệ điều hành đầu tiên dành cho máy tính cá nhân ra đời như COMMANDOR, APPLE II, IBM PC XT,… Ngoài ra, từ đầu những năm 90 cũng đánh dấu sự phát triển mạnh mẽ khác của hệ điều hành mạng và hệ điều hành phân tán.

1.3 Vai trò của hệ điều hành

- Phụ thuộc vào quan điểm (người dùng, hệ thống)

- Người dùng muốn tiện lợi, dễ sử dụng và làm việc hiệu quả. Không quan tâm đến tài nguyên sử dụng

- Đối với các máy tính như mạinframe hoặc minicomputer, phải được thiết kế sao cho thỏa mãn tất cả người dùng truy cập

- Với người sử dụng hệ thống workstations - servers, hệ điều hành được thiết kế để giải quyết hợp lý việc sử dụng tài nguyến giữa máy trạm và máy chủ

- Đối với các máy tính xách tay, tài nguyên hệ thống ít, hệ điều hành được thiết kế nhằm tối ưu khả năng sử dụng và thời gian hoạt động của Pin

- Một số máy tính có ít hoặc không có giao diện người dùng, chẳng hạn như máy tính nhúng trong các thiết bị gia dụng và xe ô tô, hệ điều hành được thiết kế để chạy không có sự can thiệp của người sử dụng

CHƯƠNG 2: TẮC NGHẼN TRONG HỆ ĐIỀU HÀNH

2.1 Tài nguyên

- Tài nguyên hệ thống là bất kỳ phần nào mà máy tính có thể sử dụng được để kiểm soát và gán bởi hệ điều hành để tất cả phần cứng và phần mềm trên máy tính có thể cùng hoạt động với nhau như thiết kế ban đầu.

2.2 Tắc nghẽn

2.2.1 Định nghĩa tắc nghẽn

- 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.

2.2.2 Minh họa tắc nghẽn

- Ví dụ tắc nghẽn khi qua cầu

+ Mỗi đoạn của cây cầu được xem là tài nguyên

+ Những xe ô tô là các tiến trình

+ Trên cây cầu hẹp chỉ vừa 1 chiếc xe đi qua, hai hay nhiều ô tô cùng đi qua sẽ đối đầu nhau dẫn đến tắc nghẽn không xe nào qua được

+ Nếu tắc nghẽn xuất hiện thì có thể được giải quyết bằng cách một hoặc một số ô tô lùi lại nhường đường và đi qua sau

2.3 Điều kiện xuất hiện tắc nghẽn

- Deadlock có thể xảy ra nếu 4 điều kiện sau tồn tại:

+ Độc quyền sử dụng (mutual exclusion): ít nhất một tài nguyên được giữ ở dạng không chia sẻ chế độ, tại 1 thời điểm chỉ có 1 tiến trình sử dụng tài nguyên nếu tiến trình khác yêu cầu thì trì hoãn cho đến khi tài nguyên đó được giải phóng (ví dụ: máy in, tệp chỉ đọc)

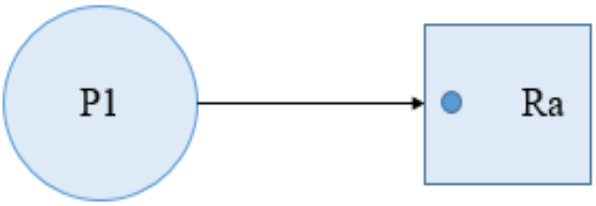
+ Giữ và đợi (hold and wait): 1 tiến trình đang sở hữu tài nguyên đã được cấp phép đang chờ thêm tài nguyên của tiến trình khác

+ Không có ưu tiên (no preemtion): 1 tài nguyên chỉ có thể được tiến trình (tự nguyện) giải phóng khi tiến trình đó đã hoàn thành nhiệm vụ của mình

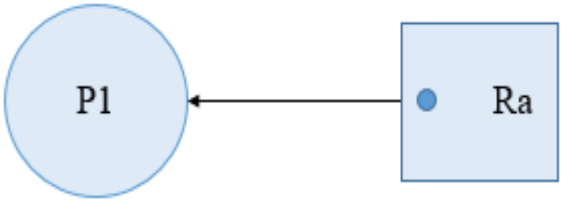
+ Chờ đợi vòng tròn (circular wait): tồn tại 1 chu kỳ đóng các yêu cầu tài nguyên. Tập hợp {P0, P1, ..., Pn} gồm các chuỗi chờ P_0 chờ tài nguyên P_1 nắm giữ, P_1 chờ tài nguyên P_2 nắm giữ,… P_n chờ tài nguyên P_0 nắm giữ

2.4 Đồ thị cấp phát tài nguyên

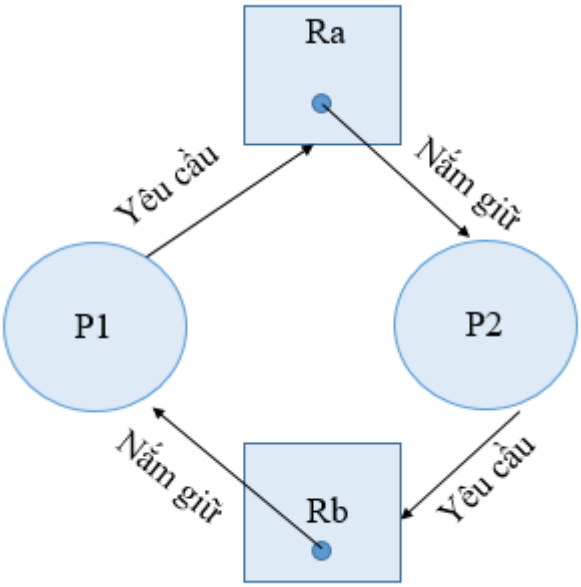
- Có thể sử dụng một đồ thị để mô hình hóa việc cấp phát tài nguyên. Đồ thị này có 2 loại nút: các tiến trình được biểu diễn bằng hình tròn, và mỗi tài nguyên được hiển thị bằng hình vuông.



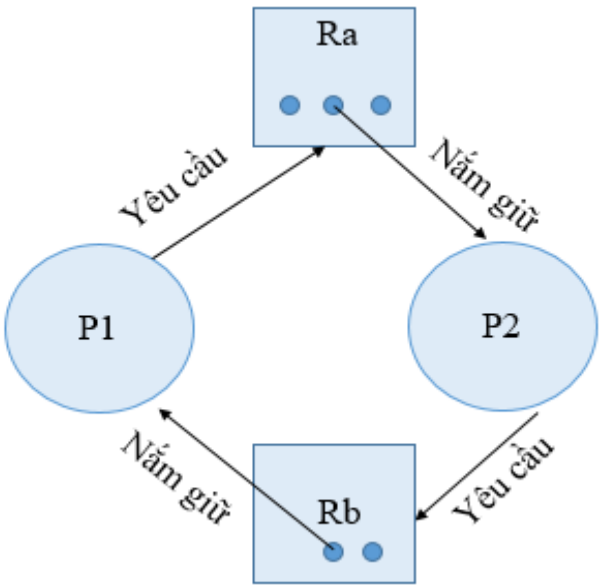
(a) Tiến trình yêu cầu tài nguyên



(b) Tiến trình nắm giữ tài nguyên



(c) Chờ đợi vòng tròn (tắc nghẽn)



(d) Không tắc nghẽn

Hình 2.2 Đồ thị cấp phát tài nguyên

2.5 Các phương pháp giải quyết tắc nghẽn

2.5.1 Tránh tắc nghẽn

Ngăn cản tắc nghẽn là một mối bận tâm lớn khi sử dụng tài nguyên. Tránh tắc nghẽn là loại bỏ tất cả các cơ hội có thể dẫn đến tắc nghẽn trong tương lai. Cần phải sử dụng những cơ chế phức tạp để thực hiện ý định này

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 <P_1, P_2,,…, P_n,> là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình P_i yêu cầu tài nguyên của P_i 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 P_j 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.

2.5.2 Ngăn chặn tắc nghẽn

- Ngăn cản lẫn nhau (mutual exclusion**):** Tài nguyên có thể chia sẻ không được yêu cầu quyền truy cập loại trừ lẫn nhau, ví dụ: tệp chỉ đọc, tài nguyên có thể chia sẻ , phải nắm giữ đối với tài nguyên không thể chia sẻ

- Giữ và đợi (hold and wait):

  + Cách 1: Mỗi tiến trình yêu cầu toàn bộ tài nguyên cần thiết một lần. Nếu có đủ tài nguyên thì hệ thống sẽ cấp phát, nếu không đủ tài nguyên thì tiến trình phải bị chặn

  + Cách 2: Khi yêu cầu tài nguyên, tiến trình không được giữ bất kỳ tài nguyên nào. Nếu đang có thì phải trả lại trước khi yêu cầu.

- Độc quyền (no preemption): Nếu tiến trình nắm giữ tài nguyên và đang yêu cầu tài nguyên khác không được cấp phát thì:

  + Cách 1: Tiến trình đó phải giải phóng các tài nguyên ban đầu của nó. Tiến trình chỉ bắt đầu lại được khi có được các tài nguyên đã bị lấy lại cùng với tài nguyên đang yêu cầu

+ Cách 2: Hệ thống sẽ xem tài nguyên mà tiến trình đang yêu cầu : nếu tài nguyên được giữ bởi một tiến trình khác đang đợi thêm tài nguyên, tài nguyên này được hệ thống lấy lại và cấp phát cho tiến trình đang yêu cầu. Nếu tài nguyên được giữ bởi tiến trình không đợi tài nguyên, tiến trình yêu cầu phải đợi và tài nguyên của tiến trình yêu cầu bị lấy lại

- Đợi vòng tròn (circular wait):Gán một thứ tự cho tất cả các tài nguyên trong hệ thống. Mỗi tiến trình chỉ có thể yêu cầu thực thể của một loại tài nguyên theo thứ tự tăng dần

Vd: Chuỗi yêu cầu thực thể hợp lệ: tape drive →disk drive →printer

Chuỗi yêu cầu thực thể không hợp lệ: disk drive → tape drive

CHƯƠNG 3: GIẢI THUẬT TRÁNH TẮC NGHẼN BANKER

3.1 Giải thuật Banker

Mỗi loại tài nguyên có nhiều thực thể giải thuật đồ thị cấp phát tài nguyên không thể áp dụng tới hệ thống cấp phát tài nguyên. Giải thuật tránh tắc nghẽn 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.

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

3.2 Cấu trúc dữ liệu giải thuật Banker

- Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker. Gọi n là số tiến trình trong hệ thống và m là số loại tài nguyên trong hệ thống. Chúng ta cần các cấu trúc dữ liệu sau:

+ Available: một vector có chiều dài m hiển thị số lượng tài nguyên sẵn có của mỗi loại. Nếu Available[j]= k, có k thực thể của loại tài nguyên R_j sẵn có

+ Max: một ma trận n × m định nghĩa số lượng tối đa yêu cầu của mỗi tiến trình. Nếu Max[i , j] = k, thì tiến trình P_i có thể yêu cầu nhiều nhất k thực thể của loại tài nguyên R_j

+ Allocation: một ma trận n × m định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp tới mỗi tiến trình. Nếu Allocation[i, j] = k, thì quá trình P_i hiện được cấp k thực thể của loại tài nguyên R_j

+ Need: một ma trận n × m hiển thị yêu cầu tài nguyên còn lại của mỗi tiến trình. Nếu Need[i, j] = k, thì tiến trình P_i có thể cần thêm k thực thể của loại tài nguyên R_j để hoàn thành nhiệm vụ của nó.

- Ta có: Need[i, j] = Max[i, j] – Allocation [i, j].

Cấu trúc dữ liệu này biến đổi theo thời gian về kích thước và giá trị

3.3 Giải thuật tìm chuỗi an toàn Banker

1. Gọi Work và Finish là các vector có chiều dài m và n tương ứng. Khởi tạo:

   Work =Available và Finish[i]=false cho i = 1, 2, …,n.

2. Tìm i thỏa:

   Finish[i] = false

   Need i ≤ Work

   Nếu không có i nào thỏa, di chuyển tới bước 4

3. Work =Work + Allocation_i

   Finish[i] = true

   Quay lại bước 2

4. Nếu Finish[i] = true cho tất cả i, thì hệ thống đang ở trạng thái an toàn

Giải thuật này có thể yêu cầu độ phức tạp m × n^2 thao tác để quyết định trạng thái là an toàn hay không

3.4 Giải thuật yêu cầu cấp phát tài nguyên

- Cho Request_i là vector yêu cầu cho tiến trình P_i. Nếu $\text{Request}_i[j] = k$, thì quá trình P_i muốn k thực thể của loại tài nguyên R_j . Khi một yêu cầu tài nguyên được thực hiện bởi quá trình P_i, thì các hoạt động sau được thực hiện:

1. Nếu $\text{Request}_i \leq \text{Need}_i$, di chuyển tới bước 2

Ngược lại, phát sinh một điều kiện lỗi vì tiến trình vượt quá yêu cầu tối đa của nó

2. Nếu $\text{Request}_i \leq$ Available, di chuyển tới bước 3

Ngược lại, P_i phải chờ vì tài nguyên không sẵn có

3. Giả sử hệ thống cấp phát các tài nguyên được yêu cầu tới tiến trình P_i bằng cách thay đổi trạng thái sau:

   Available = Available – Request_i;

   Allocation_i= Allocation_i+ Request_i;

   Need_i= Need_i– Request_i;

Nếu kết quả trạng thái cấp phát tài nguyên là an toàn thì tiến trình P_i được cấp phát tài nguyên của nó. Tuy nhiên, nếu trạng thái mới là không an toàn, thì P_i phải chờ Request_i và trạng thái cấp phát tài nguyên cũ được phục hồi

3.5 Ví dụ giải thuật Banker

Cho 5 tiến trình từ P0 – P4

3 loại tài nguyên A(10 thực thể), B(6 thực thể), C(7 thực thể)

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	1	1	2	4	3	3	2	1	0
P1	2	1	2	3	2	2			
P2	4	0	1	9	0	2			
P3	0	2	0	7	5	3			
P4	1	1	2	1	1	2			

Nội dung ma trận Need được xác định là Need = Max – Allocation:

	Need		
	A	B	C
P0	3	2	1
P1	1	1	0
P2	5	0	1
P3	7	3	3
P4	0	0	0

Chúng ta khẳng định rằng hệ thống hiện ở trong trạng thái an toàn. Thật vậy, thứ tự thỏa tiêu chuẩn an toàn.

- Đối với tiến trình P0, Need = (3, 2, 1) và Available = (2, 1, 0)

  Need > Available => False

⇨ Hệ thống sẽ chuyển sang tiến trình tiếp theo

- Đối với tiến trình P1, Need = (1,1,0) và Available = (2,1,0)

  Need < Available => True

⇨ Yêu cầu của P1 được cấp

Available = Available + Allocation

= (2, 1, 0) + (2, 1, 2)

= (4, 2, 2)

- Đối với tiến trình P2, Need = (5, 0, 1) và Available = (4, 2, 2)

Need > Available => False

⇨ Hệ thống sẽ chuyển sang tiến trình tiếp theo

- Đối với tiến trình P3, Need = (7, 3, 3) và Available = (4, 2, 2)

Need > Available => False

⇨ Hệ thống sẽ chuyển sang quy trình tiếp theo.

- Đối với tiến trình P4, Need = (0, 0, 0) và Available = (4, 2, 2)

Need < Available => True

⇨ Yêu cầu của P4 được cấp

Available = Available + Allocation

= (4, 2, 2) + (1, 1, 2)

= (5, 3, 4)

- Kiểm tra tiến trình P2, Need = (5, 0, 1) và Available = (5, 3, 4)

Need < Available => True

⇨ Yêu cầu của P2 được cấp

Available = Available + Allocation

= (5, 3, 4) + (4, 0, 1)

= (9, 3, 5)

- Kiểm tra tiến trình P3, Need = (7, 3, 3) và Available = (9, 3, 5)

Need < Available => True

⇨ Yêu cầu của P3 được cấp

Available = Available +Allocation

= (9, 3, 5) + (0, 2, 0)

= (9, 5, 5)

- Kiểm tra Quy trình P0, Need (3, 2, 1) và Available (9, 5, 5)

Need < Available => True

⇨ Yêu cầu sẽ được cấp cho P0.

Chuỗi an toàn: <P1, P4, P2, P3, P0>

CHƯƠNG 4: GIẢI THUẬT PHÁT HIỆN TẮC NGHẼN

4.1 Phát hiện tắc nghẽn

Tắc nghẽn tồn tại khi và chỉ khi có các tiến trình chưa được đánh dấu ở cuối thuật toán. Mỗi tiến trình không được đánh dấu thì bị tắc nghẽn. Thuật toán này là tìm một tiến trình mà yêu cầu tài nguyên có thể thỏa mãn với các tài nguyên có sẵn sau đó giả sử các nguồn đó được cấp và nếu tiến trình chạy đến hoàn thành và giải phóng tất cả tài nguyên của nó. Thuật toán sau đó tìm kiếm 1 tiến trình khác để thõa mãn.

Nếu một hệ thống không sử dụng ngăn chặn tắc nghẽn hoặc tránh tắc nghẽn khi đó tình huống tắc nghẽn có thể xảy ra. Trong môi trường này, hệ thống có thể cung cấp:

- Một thuật toán kiểm tra trạng thái của hệ thống để xác định xem một tắc nghẽn đã xảy ra
- Một thuật toán để khôi phục từ bế tắc

4.2 Mỗi loại tài nguyên có nhiều thực thể

Đồ thị chờ đợi không áp dụng được cho hệ thống phân bổ tài nguyên với mỗi loại tài nguyên có nhiều thực thể. Chúng ta sử dụng một thuật toán phát hiện tắc nghẽn có thể áp dụng cho một hệ thống, thuật toán sử dụng một số cấu trúc dữ liệu thay đổi theo thời gian như:

+ Available: một vectơ có độ dài m cho biết số lượng tài nguyên hiện có của mỗi loại.

+ Allocation: một ma trận n × m xác định số lượng tài nguyên của mỗi loại hiện được phân bổ cho mỗi tiến trình

+ Request: một ma trận n × m, cho biết yêu cầu của mỗi tài nguyên hiện tại. Nếu Request [i][j] bằng k, thì tiến trình P_i đang yêu cầu k thực thể khác của loại tài nguyên R_j

4.3 Mỗi loại tài nguyên có một thực thể

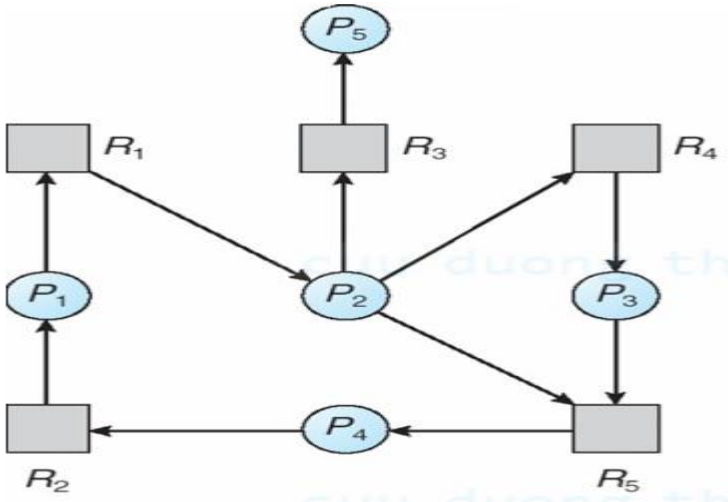
- Nếu tất cả các tài nguyên chỉ có một thực thể duy nhất, thì chúng ta có thể xác định thuật toán phát hiện tắc nghẽn sử dụng một biến thể của đồ thị phân bổ tài nguyên, được gọi là đồ thị chờ. Chúng ta thu được đồ thị này từ đồ thị phân bổ tài nguyên bằng cách loại bỏ các nút tài nguyên và thu gọn các cạnh thích hợp

- Đồ thị chờ

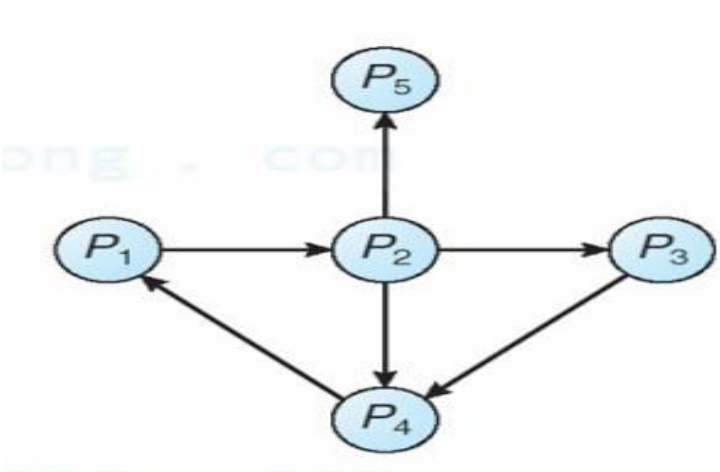
+ Đỉnh là các tiến trình

+ Một cạnh từ P_i đến P_j ($P_i \rightarrow P_j$) trong đồ thị chờ cho biết rằng tiến trình P_i đang chờ tiến trình P_j giải phóng 1 tài nguyên mà P_i cần.

+ Một cạnh $P_i \rightarrow P_j$ tồn tại trong đồ thị chờ khi và chỉ khi đồ thị phân bổ tài nguyên tương ứng chứa 2 cạnh $P_i \rightarrow P_q$ và $P_q \rightarrow P_j$. Trong đó P_q là một số tài nguyên



Hình 4.1 Đồ thị cấp phát tài nguyên



Hình 4.2 Đồ thị chờ

Trong một hệ thống, tắc nghẽn tồn tại nếu đồ thị chờ chứa một chu trình. Để phát hiện được tắc nghẽn gọi thuật toán tìm kiếm một chu trình trong đồ thị. Một thuật

toán để phát hiện chu trình trong đồ thị thì cần thực hiện n^2 các thao tác lệnh. Trong đó, n là số đỉnh của đồ thị

4.4 Giải thuật phát hiện tắc nghẽn

1. Gọi Work và Finish là các vecto với độ dài tương ứng m và n, được khởi tạo:

   Work = Available

   Với chỉ số i từ i = 1, 2, 3,…,n, nếu $\text{Allocation}_i \neq 0$ thì Finish[i] = false. Ngược lại, Finish[i] = true

2. Tìm chỉ số i sao cho thoả 2 điều kiện:

   Finish[i] == false

   $\text{Request}_i \leq \text{Work}$

   Nếu i không tồn tại thì sang bước 4

3. Work = Work + Allocation_i

   Finish[i] = true

   Quay lại bước 2

4. Nếu Finish[i] == false với một số i, $1 \leq i \leq n$, thì hệ thống ở trạng thái tắc nghẽn. Nếu Finish[i] == false thì tiến trình P_i bị tắc nghẽn

4.4 Ví dụ giải thuật tắc nghẽn

Cho 5 tiến trình từ P0 – P4

Loại tài nguyên A(7 thực thể), B(2 thực thể), C(6 thực thể)

	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			

Tại thời điểm T_0 chúng ta có trạng thái cấp phát tài nguyên sau:

- Đối với tiến trình P0, Request = (0,0,0) và Available = (0,0,0)

Request ≤ Available => True

⇨ Yêu cầu P0 được cấp phát

Available = Available + Allocation

= (0,0,0) + (0,1,0)

= (0,1,0)

- Đối với tiến trình P1, Request = (2,0,2) và Available = (0,1,0)

Request > Available => False

⇨ Hệ thống chuyển sang tiến trình tiếp theo

- Đối với tiến trình P2, Request = (0,0,0) và Available = (0,1,0)

Request ≤ Available => True

⇨ Yêu cầu P2 được cấp phát

Available = Available + Allocation

= (0,1,0)+ (3,0,3)

= (3,1,3)

- Đối với tiến trình P3, Request = (1,0,0) và Available = (3,1,3)

Request < Available => True

⇨ Yêu cầu P3 được cấp phát

Available = Available + Allocation

= (3,1,3)+ (2,1,1)

= (5,2,4)

- Kiểm tra tiến trình P1, Request = (2,0,2) và Available = (5,2,6)

Request < Available => True

⇨ Yêu cầu P1 được cấp phát

Available = Available + Allocation

= (2,0,0) + (5,2,4)

= (7,2,4)

- Đối với tiến trình P4, Request = (0,0,2) và Available = (7,2,4)

Request < Available => True

⇨ Yêu cầu P4 được cấp phát

Available = Available + Allocation

= (0,0,2) + (7,2,4)

= (7,2,6)

Finish[i] = True với mọi i, vì vậy có chuỗi an toàn <P0, P2, P3, P1, P4> => hệ thống an toàn

Nếu P2 đòi thêm 1 thực thể của tài nguyên C

	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	1			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Tại thời điểm T_0 chúng ta có trạng thái cấp phát tài nguyên sau:

- Đối với tiến trình P0, Request = (0,0,0) và Available = (0,0,0)

  Request ≤ Available => True

⇨ Yêu cầu P0 được cấp phát

Available = Available + Allocation

= (0,0,0) + (0,1,0)

= (0,1,0)

- Đối với tiến trình P1, Request = (2,0,2) và Available = (0,1,0)

  Request > Available => False

⇨ Hệ thống chuyển sang tiến trình tiếp theo

- Đối với tiến trình P2, Request = (0,0,1) và Available = (0,1,0)

  Request > Available => False

⇨ Hệ thống chuyển sang tiến trình tiếp theo

- Đối với tiến trình P3, Request = (1,0,0) và Available = (0,1,0)

  Request > Available => False

⇨ Hệ thống chuyển sang tiến trình tiếp theo

- Đối với tiến trình P4, Request = (0,0,2) và Available = (0,1,0)

Request > Available => False

Chỉ có thể cấp được tài nguyên cho tiến trình P0 và đòi lại được tài nguyên từ P0, nhưng không đủ tài nguyên cho tiến trìn P1, P2, P3 và P4. Vì vậy, hệ thống bị tắc nghẽn gồm các tiến trình P1, P2, P3 và P4

4.5 Khôi phục tắc nghẽn

Khi thuật toán phát hiện tắc nghẽn đã xác định được có tắc nghẽn tồn tại sẽ thông báo cho người vận hành tắc nghẽn đã xảy ra và để người vận hành xử lý tắc nghẽn. Mặt khác, hệ thống tự động khôi phục từ tắc nghẽn

4.5.1 Khôi phục tắc nghẽn: kết thúc tiến trình

- Hủy bỏ tất cả các tiến trình bị tắc nghẽn: đây là một trong những giải pháp phổ biến nhất. Các tiến trình bị tắc nghẽn đã được thực hiện trong một thời gian dài, kết quả của những tính toán từng phần này phải bị loại bỏ và sẽ tính toán lại sau này.

- Huỷ bỏ từng tiến trình một cho đến khi loại bỏ được chu trình tắc nghẽn: Thứ tự các tiến trình được lựa chọn để hủy bỏ phải dựa trên một số tiêu chí về chi phí tối thiểu. Sau khi tiến trình bị hủy bỏ, thuật toán phát hiện tắc nghẽn được sử dụng để xác định xem tắc nghẽn còn tồn tại không

4.5.2 Khôi phục tắc nghẽn: sự ưu tiên tài nguyên

- Lựa chọn tiến trình bị thu hồi tài nguyên: chúng ta phải xác định thứ tự ưu tiên để giảm thiểu chi phí. Chi phí có thể bao gồm là số lượng tài nguyên mà tiến trình tắc nghẽn đang giữ và lượng thời gian mà tiến trình đã tiêu tốn

- Rollback: nếu chúng ta loại trừ một tài nguyên khỏi một tiến trình thì nó có thể không thực hiện được tiếp tục vì nó đang thiếu một số tài nguyên cần thiết. Vì thế, phải khôi phục về trạng thái an toàn và khởi động lại tiến trình ở trạng thái đó

- Starvation: việc lựa chọn tiến trình bị thu hồi tài nguyên dựa trên các yếu tố chi phí, cũng có thể xảy ra tiến trình luôn được chọn là tiến trình bị thu hồi tài nguyên. Dẫn đến tiến trình này không bao giờ hoàn thành nhiệm vụ của nó. Vì vậy, chúng ta phải đảm bảo rằng một tiến trình có thể được chọn làm tiến trình bị thu hồi tài nguyên chỉ một số lần nhỏ hữu hạn

CHƯƠNG 5: CHƯƠNG TRÌNH GIẢI THUẬT BANKER VÀ PHÁT HIỆN TẮC NGHẼN

5.1 Giải thuật Banker

```
#include <stdio.h>
#include <conio.h>
void main()
{
	int dem = 0, m, n, tien_trinh, tam, tai_nguyen;
	int bang_tai_nguyen[5] = { 0, 0, 0, 0, 0 };
	int tai_nguyen_co_san[5], tai_nguyen_hien_hanh[5][5];
	int yeu_cau_toi_da[5][5];
	int tai_nguyen_toi_da[5], dang_chay[5], trang_thai_an_toan = 0;
	printf("\n**********TRANH TAC NGHEN BANKER**********");
	printf("\n*                    NHOM 10                    *");
	printf("\n************************************************\n");
	printf("\nNhap so tien trinh: ");
	scanf("%d", &tien_trinh);
	for (m = 0; m < tien_trinh; m++)
	{
		dang_chay[m] = 1;
		dem++;
	}
	printf("\nNhap tong so loai tai nguyen cap phat: ");
	scanf("%d", &tai_nguyen);
	printf("\nNhap tong so tai nguyen moi loai: ");
	for (m = 0; m < tai_nguyen; m++)
	{
		scanf("%d", &tai_nguyen_toi_da[m]);
	}
	printf("\nNhap ma tran tai nguyen da cap phat (moi tien trinh 1 dong)\n");
```

```
for (m = 0; m < tien_trinh; m++)
{
	for (n = 0; n < tai_nguyen; n++)
	{
		scanf("%d", &tai_nguyen_hien_hanh[m][n]);
	}
}
printf("\nNhap ma tran tai nguyen yeu cau toi da cua cac tien trinh (moi tien trinh 1 dong)\n");
for (m = 0; m < tien_trinh; m++)
{
	for (n = 0; n < tai_nguyen; n++)
	{
		scanf("%d", &yeu_cau_toi_da[m][n]);
	}
}
printf("\tKET QUA GIAI THUAT BANKER\n");
printf("\nTong moi loai tai nguyen cua he thong\n");
for (m = 0; m < tai_nguyen; m++)
{
	printf("\t%d", tai_nguyen_toi_da[m]);
}
printf("\nMa tran tai nguyen da cap phat\n");
for (m = 0; m < tien_trinh; m++)
{
	for (n = 0; n < tai_nguyen; n++)
	{
		printf("\t%d", tai_nguyen_hien_hanh[m][n]);
	}
	printf("\n");
```

```
}
printf("\nMa tran tai nguyen toi da ma cac tien trinh can dung\n");
for (m = 0; m < tien_trinh; m++)
{
        for (n = 0; n < tai_nguyen; n++)
        {
                printf("\t%d", yeu_cau_toi_da[m][n]);
        }
        printf("\n");
}
for (m = 0; m < tien_trinh; m++)
{
        for (n = 0; n < tai_nguyen; n++)
        {
                bang_tai_nguyen[n] += tai_nguyen_hien_hanh[m][n];
        }
}
printf("\nTai nguyen da cap phat\n");
for (m = 0; m < tai_nguyen; m++)
{
        printf("\t%d", bang_tai_nguyen[m]);
}
for (m = 0; m < tai_nguyen; m++)
{
       tai_nguyen_co_san[m] = tai_nguyen_toi_da[m] - bang_tai_nguyen[m];
}
printf("\nTai nguyen co san cua he thong\n");
for (m = 0; m < tai_nguyen; m++)
{
        printf("\t%d", tai_nguyen_co_san[m]);
```

```
}
printf("\n");
printf("\nTIEN TRINH\t\tCHUOI AN TOAN\t\t TAI NGUYEN SAN CO\n");
while (dem != 0)
{
        trang_thai_an_toan = 0;
        for (m = 0; m < tien_trinh; m++)
        {
                if (dang_chay[m])
                {
                        tam = 1;
                        for (n = 0; n < tai_nguyen; n++)
                        {
                        if  (yeu_cau_toi_da[m][n] –tai_nguyen_hien_hanh[m][n]
                                > tai_nguyen_co_san[n])
                                {
                                        tam = 0;
                                        break;
                                }
                        }
                        if (tam)
                        {
                                printf("\nTIEN TRINH P%d\t", m);
                                dang_chay[m] = 0;
                                dem--;
                                trang_thai_an_toan = 1;
                                for (n = 0; n < tai_nguyen; n++)
                                {
                                        tai_nguyen_co_san[n] +=
```

```
                                        tai_nguyen_hien_hanh[m][n];
                                }
                                break;
                        }
                }
        }
        if (!trang_thai_an_toan)
        {
                printf("\tTien trinh o trang thai khong an toan\n");
                break;
        }
        else
        {
                printf("\tTien trinh an toan");
                for (m = 0; m < tai_nguyen; m++)
                {
                        printf("\t%d", tai_nguyen_co_san[m]);
                }
                printf("\n");
        }
    }
    getch();
}
```

- Kết quả chạy chương trình

```
**************************************TRANH TAC NGHEN BANKER*************************************
*                                          NHOM 10                                              *
*************************************************************************************************

Nhap so tien trinh: 5

Nhap tong so loai tai nguyen cap phat: 3

Nhap tong so tai nguyen moi loai: 10 6 7

Nhap ma tran tai nguyen da cap phat (moi tien trinh 1 dong)
1 1 2
2 1 2
4 0 1
0 2 0
1 1 2

Nhap ma tran tai nguyen yeu cau toi da cua cac tien trinh (moi tien trinh 1 dong)
4 3 3
3 2 2
9 0 2
7 5 3
1 1 2
       KET QUA GIAI THUAT BANKER

Tong moi loai tai nguyen cua he thong
       10     6      7
```

```
Ma tran tai nguyen da cap phat
        1       1       2
        2       1       2
        4       0       1
        0       2       0
        1       1       2

Ma tran tai nguyen toi da ma cac tien trinh can dung
        4       3       3
        3       2       2
        9       0       2
        7       5       3
        1       1       2

Tai nguyen da cap phat
        8       5       7
Tai nguyen co san cua he thong
        2       1       0

TIEN TRINH              CHUOI AN TOAN            TAI NGUYEN SAN CO

TIEN TRINH P1           Tien trinh an toan      4       2       2

TIEN TRINH P0           Tien trinh an toan      5       3       4

TIEN TRINH P2           Tien trinh an toan      9       3       5

TIEN TRINH P3           Tien trinh an toan      9       5       5

TIEN TRINH P4           Tien trinh an toan      10      6       7
```

5.2 Giải thuật phát hiện tắc nghẽn

#include<stdio.h>

#include<conio.h>

```
void main()
{
	int yeu_cau[100][100];
	int tai_nguyen_hien_hanh[100][100];
	int yeu_cau_tai_nguyen[100][100];
	int tai_nguyen_san_co[100];
	int n, r;
	int i, j;
	int finish[100], temp,flag = 1, k, c1 = 0;
	int tac_nghen[100];
	int trang_thai_an_toan[100];
	printf("\n**************************************************");
	printf("\n*                    NHOM 10                     *");
	printf("\n*          GIAI THUAT PHAT HIEN TAC NGHEN        *");
	printf("\n**************************************************\n")
	printf("Nhap so tien trinh: ");
	scanf("%d", &n);
	printf("\nNhap so loai tai nguyen cap phat: ");
	scanf("%d", &r);
	printf("\nNhap ma tran tai nguyen yeu cau hien tai (Request) cua moi tien trinh
	(moi tien trinh 1 dong)\n");
	for (i = 0; i<n; i++)
	{
		for (j = 0; j<r; j++)
		{
			scanf("%d", &yeu_cau[i][j]);
		}
	}
	printf("\nNhap ma tran tai nguyen da cap phat(Allocation) (moi tien trinh 1
	dong)\n");
```

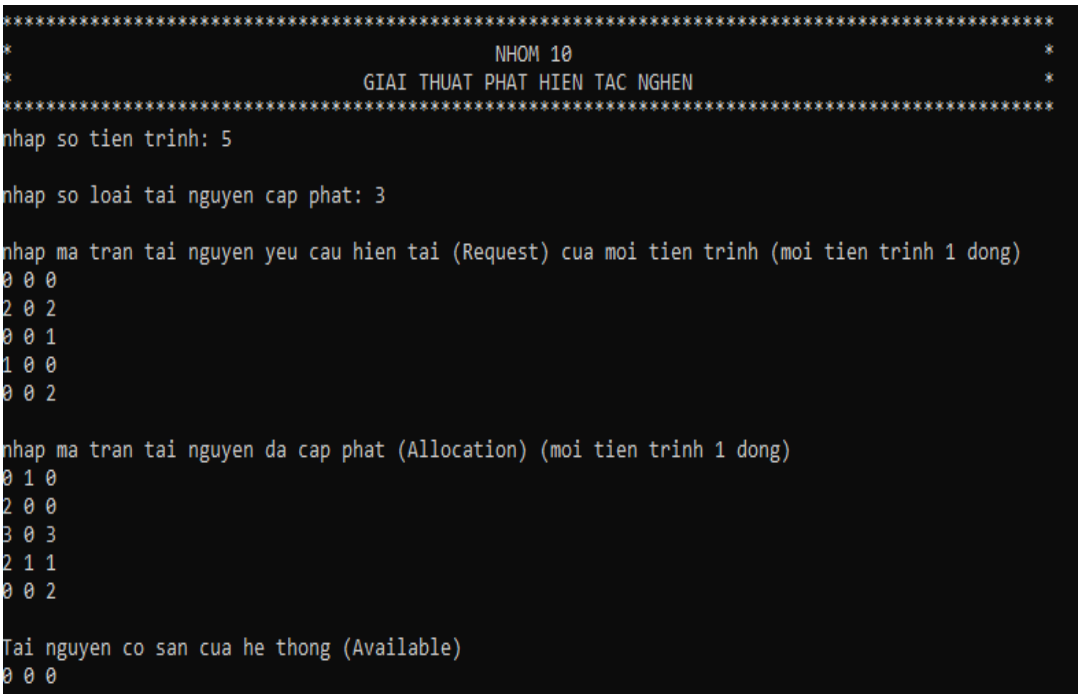
```
for (i = 0; i<n; i++)
{
	for (j = 0; j<r; j++)
	{
		scanf("%d", &tai_nguyen_hien_hanh[i][j]);
	}
}
printf("\nTai nguyen co san cua he thong(Available)\n");
for (j = 0; j<r; j++)
{
	scanf("%d", &tai_nguyen_san_co[j]);
}
printf("\tKET QUA GIAI THUAT PHAT HIEN TAC NGHEN\n");
printf("\nTIEN TRINH\t Allocation\tRequest\t Available\t");
for (i = 0; i<n; i++)
{
	printf("\nTien trinh P%d\t   ", i);
	for (j = 0; j<r; j++)
	{
		printf("%d ", tai_nguyen_hien_hanh[i][j]);
	}
	printf("\t");
	for (j = 0; j<r; j++)
	{
		printf("%d ", yeu_cau[i][j]);
	}
	printf("\t");
	if (i == 0)
	{
		for (j = 0; j<r; j++)
```

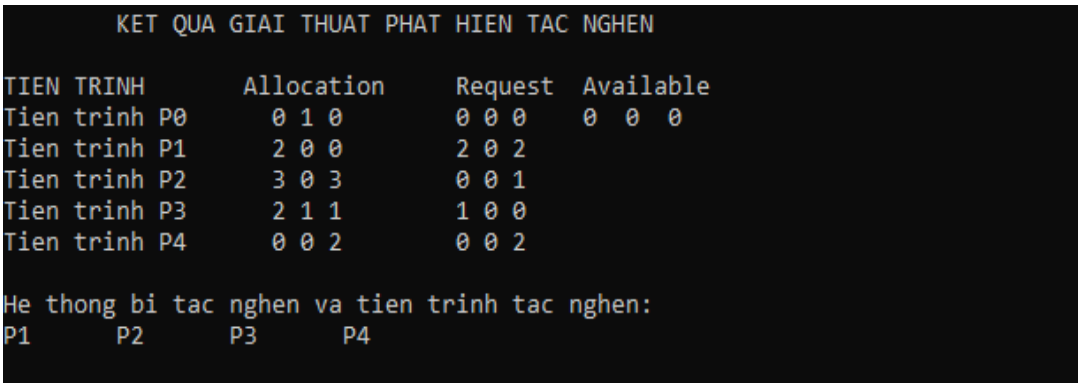
```
                printf(" %d ", tai_nguyen_san_co[j]);
        }
}
for (i = 0; i<n; i++)
{
        finish[i] = 0;
}
for (i = 0; i<n; i++)
{
        for (j = 0; j<r; j++)
        {
                yeu_cau_tai_nguyen[i][j] = yeu_cau[i][j] ;
        }
}
while (flag)
{
        flag = 0;
        for (i = 0; i<n; i++)
        {
                int c = 0;
                for (j = 0; j<r; j++)
                {
                        if ((finish[i] == 0) && (yeu_cau_tai_nguyen[i][j] <=
                        tai_nguyen_san_co[j]))
                        {
                                c++;
                                if (c == r)
                                {
                                        for (k = 0; k<r; k++)
                                        {
```

```
                                        tai_nguyen_san_co[k]+=
                                        tai_nguyen_hien_hanh[i][j];
                                        finish[i] = 1;
                                        flag = 1;
                                }
                                if (finish[i] == 1)
                                {
                                        i = n;
                                }
                        }
                }
        }
}
j = 0;
flag = 0;
for (i = 0; i<n; i++)
{
        if (finish[i] == 0)
        {
                tac_nghen[j] = i;
                j++;
                flag = 1;
        }
}
if (flag == 1)
{
        printf("\n\nHe thong bi tac nghen va tien trinh tac nghen:\n");
        for (i = 0; i<n-1; i++)
        {
```

```
                printf("P%d\t", tac_nghen[i]);
            }
        }
        else
        {
            printf("\n\n\tHe thong khong bi tac nghen!!!");
        }
        getch();
}
```

- Kết quả chạy chương trình





KẾT LUẬN

Qua bài tiểu luận tìm hiểu về giải thuật quản lý tắc nghẽn trong hệ điều hành và viết chương trình minh họa nhóm chúng em đã rút ra được trong môi trường đa chương một số tiến trình có thể cạnh tranh để giành một số lượng tài nguyên có thể dẫn đến tắc nghẽn. Việc quản lý tắc nghẽn là vô cùng quan trọng giúp hạn chế được xảy ra tắc nghẽn, để không tiến trình nào phải chờ đợi mãi. Giải thuật Banker giúp hệ thống tìm ra chuỗi an toàn để các tiến trình đều được cấp phát tài nguyên và giải phóng tài nguyên cho các tiến trình tiếp theo. Giải thuật phát hiện tắc nghẽn giúp hệ thống phát hiện ra tiến trình nào bị tắc nghẽn để người vận hành xử lý tắc nghẽn hoặc là hệ thống tự động khôi phục từ tắc nghẽn

TÀI LIỆU THAM KHẢO

[1] Andrew S.Tanenbaum, Allbert S.Woodhull. *Operating Systems Design & Implementaiton*

[2] *Các thành phần hệ điều hành, nhân hệ điều hành, tải hệ điều hàn*h https://hocday.com/cu-1-cc-thnh-phn-ca-h-iu-hnh-nhn-h-iu-hnh-ti-h-iu-hnh.html

[3] Chương 7. *Lý thuyết môn học hệ điều hành*

[4] *Tài liệu môn hệ điều hành chương 6 tắc nghẽn (deadlock)* http://www.tailieu.tv/tai-lieu/giao-trinh-mon-he-dieu-hanh-chuong-6-tac-nghen-deadlock-32606/

[5] William Stalling. *Operating Systems Internals and Design Principles*