TRƯỜNG ĐẠI HỌC SƯ PHẠM TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

TIỂU LUẬN MÔN HỌC
HỆ ĐIỀU HÀNH

ĐỀ TÀI:

1/ Tìm hiểu lịch sử phát triển hệ điều hành.

2/ Xây dựng chương trình minh họa giải thuật Banker để phát hiện và giải quyết Deadlock.

GVHD: TS. Trần Sơn Hải
CN. Lê Thanh Thoại

Nhóm sinh viên:

1. **Đoàn Văn Nhân – 48.01.104.099**
2. **Phạm Quốc Thoại – 48.01.104.130**
3. **Nguyễn Tấn Tú – 48.01.104.142**
4. **Tăng Quốc Khải – 48.01.104.061**
5. **Nông Thị Nhật Lệ – 48.01.104.074**

Thành phố Hồ Chí Minh – 05/2024

MỤC LỤC

BẢNG HÌNH VẼ

BẢNG PHÂN CÔNG CÔNG VIỆC

STT	MSSV	Họ và Tên	Công việc được giao	Đánh giá
1	48.01.104.099	Đoàn Văn Nhân	- Word báo cáo - Code mô phỏng thuật toán - Soạn nội dùng	100%
2	48.01.104.130	Phạm Quốc Thoại	- Word báo cáo - Code mô phỏng thuật toán - Soạn nội dùng	100%
3	48.01.104.142	Nguyễn Tấn Tú	- Code mô phỏng thuật toán - Soạn nội dùng	100%
4	48.01.104.061	Tăng Quốc Khải	- Power point - Code mô phỏng thuật toán - Soạn nội dùng	100%
5	48.01.104.074	Nông Thị Nhật Lệ	- Code mô phỏng thuật toán - Soạn nội dùng - Quay video, thuyết trình	100%

TÓM TẮT TIỂU LUẬN

Quá trình nghiên cứu và thực hiện tiểu luận về lịch sử phát triển của hệ điều hành cùng việc xây dựng chương trình minh họa thuật toán Banker, chúng tôi đã bắt đầu với việc lựa chọn ngôn ngữ lập trình phù hợp. Ban đầu, chúng tôi quyết định sử dụng C++ để xây dựng chương trình mô phỏng thuật toán. Tuy nhiên, sau khi thử nghiệm, chúng tôi nhận ra rằng giao diện của chương trình không đạt được mức độ thẩm mỹ và tương tác mà chúng tôi mong muốn. Do đó, quyết định chuyển sang sử dụng HTML, CSS và JavaScript để tạo ra một giao diện web linh hoạt và hấp dẫn hơn.

Chương trình minh họa của chúng tôi được thiết kế để giúp người dùng hiểu rõ về cách hoạt động của thuật toán Banker và là một công cụ hữu ích để thực hành và rèn luyện kỹ năng lập trình. Giao diện người dùng được thiết kế đơn giản nhưng dễ sử dụng, cho phép người dùng tương tác với chương trình một cách thuận tiện.

Sử dụng JavaScript, chúng tôi minh họa cách thuật toán Banker hoạt động thông qua việc diễn giải các bước và quá trình xử lý tài nguyên. Khi có yêu cầu tài nguyên mới từ một tiến trình, chương trình sẽ kiểm tra xem việc cấp phát tài nguyên đó có làm tình trạng deadlock hay không và thông báo kết quả cho người dùng.

Chương trình minh họa thuật toán Banker không chỉ giúp hiểu rõ về cơ chế hoạt động của thuật toán mà còn là một công cụ hữu ích để thực hành và rèn luyện kỹ năng lập trình trên môi trường web. Đây thực sự là một hành trình thú vị và đầy ý nghĩa đối với chúng tôi.

Đề tài này gồm có 2 chương:

Chương 1. Tìm hiểu về lịch sử phát triển của Hệ điều hành

Chương 2. Xây dương chương trình minh họa giải thuật Banker

Chương 1: Tìm hiểu về lịch sử phát triển của Hệ điều hành

1.1 Hệ điều hành là gì

Hệ điều hành là một chương trình hay một hệ chương trình hoạt động giữa ngườisử dụng (user) và phần cứng của máy tính. Mục tiêu của hệ điều hành là cung cấp một môi trường để người sử dụng có thể thi hành các chương trình. Nó làm cho máy tính dể sử dụng hơn, thuận lợi hơn và hiệu quả hơn.

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.

1.2 Lịch sử hệ điều hành

1.2.1 Thế hệ 1 (1945 – 1955)

Vào khoảng giữa thập niên 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 rẻ nhất ngày nay.

Mỗi máy được một nhóm thực hiện tất cả từ thiết kế, xây dựng lập trình, thao tác đến quản lý. Lập trình bằng ngôn ngữ máy tuyệt đối, thường là bằng cách dùng bảng điều khiển để thực hiện các chức năng cơ bản. Ngôn ngữ lập trình chưa được biết đến và hệ điều hành cũng chưa nghe đến.

Vào đầu thập niên 1950, phiếu đục lổ ra đời và có thể viết chương trình trên phiếu thay cho dùng bảng điều khiển.

1.2.2 Thế hệ 2 (1955 – 1965)

Sự ra đời của thiết bị bán dẫn vào giữa thập niên 1950 làm thay đổi bức tranh tổng thể. Máy tính trở nên đủ tin cậy hơn. Nó được sản xuất và cung cấp cho

các khách hàng. Lần đầu tiên có sự phân chia rõ ràng giữa người thiết kế, người xây dựng, người vận hành, người lập trình, và người bảo trì.

Để thực hiện một công việc (một chương trình hay một tập hợp các chương trình), lập trình viên trước hết viết chương trình trên giấy (bằng hợp ngữ hay FORTRAN) sau đó đục lỗ trên phiếu và cuối cùng đưa phiếu vào máy. Sau khi thực hiện xong nó sẽ xuất kết quả ra máy in.

Hệ thống xử lý theo lô ra đời, nó lưu các yêu cầu cần thực hiện lên băng từ, và hệ thống sẽ đọc và thi hành lần lượt. Sau đó, nó sẽ ghi kết quả lên băng từ xuất và cuối cùng người sử dụng sẽ đem băng từ xuất đi in.

Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của một chương trình đặc biệt là tiền thân của hệ điều hành sau này. Ngôn ngữ lập trình sử dụng trong giai đoạn này chủ yếu là FORTRAN và hợp ngữ.

1.2.3 Thế hệ 3 (1965 – 1980)

Trong giai đoạn này, máy tính được sử dụng rộng rãi trong khoa học cũng như trong thương mại. Máy IBM 360 là máy tính đầu tiên sử dụng mạch tích hợp (IC). Từ đó kích thước và giá cả của các hệ thống máy giảm đáng kể và máy tính càng phổ biến hơn. Các thiết bị ngoại vi dành cho máy xuất hiện ngày càng nhiều và thao tác điều khiển bắt đầu phức tạp.

Hệ điều hành ra đời nhằm điều phối, kiểm soát hoạt động và giải quyết các yêu cầu tranh chấp thiế bị. Chương trình hệ điều hành dài cả triệu dòng hợp ngữ và do hàng ngàn lập trình viên thực hiện.

Sau đó, hệ điều hành ra đời khái niệm *đa chương*. CPU không phải chờ thực hiện các thao tác nhập xuất. Bộ nhớ được chia làm nhiều phần, mỗi phần có một công việc (job) khác nhau, khi một công việc chờ thực hiện nhập xuất CPU sẽ xử lý các công việc còn lại. Tuy nhiên khi có nhiều công việc cùng xuất hiện trong bộ

nhớ, vấn đề là phải có một cơ chế bảo vệ tránh các công việc ảnh hưởng đến nhau. Hệ điều hành cũng cài đặt thuộc tính spool.

Giai đoạn này cũng đánh dấu sự ra đời của *hệ điều hành chia xẻ thời gian* như CTSS của MIT. Đồng thời các hệ điều hành lớn ra đời như MULTICS, UNIX và hệ thống các máy mini cũng xuất hiện như DEC PDP-1.

1.2.4 Thế hệ 4 (1980 -)

Giai đoạn này đánh dấu sự ra đời của máy tính cá nhân, đặc biệt là hệ thống IBM PC với hệ điều hành MS-DOS và Windows sau này. Bên cạnh đó là sự phát triển mạnh của các hệ điều hành tựa Unix trên nhiều hệ máy khác nhau như Linux. Ngoài ra, từ đầu thập niên 90 cũng đánh dấu sự phát triển mạnh mẽ của *hệ điều hành mạng* và *hệ điều hành phân tán*.

Chương 2: Xây dựng chương trình minh họa giải thuật Banker để phát hiện và giải quyết Deadlock.

2.1 Giới thiệu chung

2.1.1 Deadlock là gì

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

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

2.1.2 Điều kiện xảy ra Deadlock

Coffman, Elphick và Shoshani đã đưa ra 4 điều kiện cần có thể làm xuất hiện tắc nghẽn:

Có sử dụng tài nguyên không thể chia sẻ (Mutual exclusion): Mỗi thời điểm, một tài nguyên không thể chia sẻ được hệ thống cấp phát chỉ cho một tiến trình , khi tiến trình sử dụng xong tài nguyên này, hệ thống mới thu hồi và cấp phát tài nguyên cho tiến trình khác.

Sự chiếm giữ và yêu cầu thêm tài nguyên (Wait for): Các tiến trình tiếp tục chiếm giữ các tài nguyên đã cấp phát cho nó trong khi chờ được cấp phát thêm một số tài nguyên mới.

Không thu hồi tài nguyên từ tiến trình đang giữ chúng (No preemption): Tài nguyên không thể được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sủ dụng chúng xong.

Tồn tại một chu kỳ trong đồ thị cấp phát tài nguyên (Circular wait): có ít nhất hai tiến trình chờ đợi lẫn nhau : tiến trình này chờ được cấp phát tài nguyên đang bị tiến trình kia chiếm giữ và ngược lại.

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

2.1.3 Phương pháp xử lý tắt nghẽn

Có ba giải pháp cho hiện tượng Deadlock như sau:

1. Ngăn ngừa hoặc tránh xa, đảm bảo hệ thống không rơi vào trạng thái deadlock.
2. Cho phép hệ thống rơi vào trạng thái deadlock rồi sau đó khắc phục.
3. Bỏ qua tất cả các vấn đề và giả định deadlock không bao giờ xuất hiện trong hệ thống.

- Deadlock không được phát hiện, dẫn đến việc giảm hiệu suất của hệ thống. Cuối cùng, hệ thống có thể ngưng hoạt động và phải được khởi động lại.
- Phương pháp này được sử dụng trong nhiều hệ điều hành, kể cả UNIX.

2.1.3 Trạng thái an toàn (safe)

- Một trạng thái của hệ thống được gọi là an toàn khi tồn tại một chuỗi an toàn.
- Một chuỗi tiến trình <P1, P2,..., Pn> là một chuỗi an toàn nếu:

+ Với mọi i = 1,...,n yêu cầu tối đa về tài nguyên của Pi có thể được thỏa bởi:

+ Tài nguyên mà hệ thống đang có sẵn sàng.

+ Cùng với tài nguyên mà tất cả các Pj (j < i) đang giữ.

2.1.4 Ngăn chặn Deadlock

Deadlock có bốn đặc điểm được xem là điều kiện cần. Như vậy, chỉ cần ngăn không cho một trong bốn điều kiện này xuất hiện là có thể ngăn chặn deadlock.

- Ngăn độc quyền truy xuất (Mutual Exclusion)

Đối với những tài nguyên không thể chia sẻ (nonshareable resource): không làm được

Đối với kiểu tài nguyên có thể chia sẻ (shareable resource vd: read-only file và tác vụ cho phép lên file chỉ đọc): không cần thiết

Ngăn giữ và chờ (Hold and wait)

Cách 1: mỗi process yêu cầu toàn bộ tài nguyên cần thiết một lần. Nếu đủ tài nguyên thì hệ thống sẽ cấp phát, nếu không đủ tài nguyên thì process sẽ bị blocked.

Cách 2: khi yêu cầu tài nguyên, process không đang giữ bất kì tài nguyên nào. Nếu đang giữ thì phải trả lại trước khi yêu cầu.

Khuyết điểm của các cách trên:

- Hiệu suất sử dụng tài nguyên (resource utilization) thấp.
- Quá trình có thể bị chết đói (starvation).
-

Ngăn không chiếm đoạt (No Preemption): cho phép lấy lại tài nguyên đã cấp phát cho quá trình, chỉ thích hợp cho tài nguyên dễ dàng lưu và phục hồi(VD: CPU, Register), không thích hợp cho loại tài nguyên như máy in, DVD.

Một thực hiện: nếu process A có giữ tài nguyên và yêu cầu tài nguyên khác nhưng tài nguyên này chưa cấp phát ngay được thì hệ thống:

- Lấy lại mọi tài nguyên mà A đang giữ
- Ghi nhận những tài nguyên của A đã bị lấy lại và tài nguyên mà A đã yêu cầu thêm.
- Tiếp tục A khi có đủ tài nguyên cho nó.

Ngăn vòng đợi (Circular wait): áp dụng một thứ tự tuyệt đối cho tất cá các loại tài nguyên

Mỗi tiến trình yêu cầu các tài nguyên theo thứ tự tăng dần: chỉ có thể nhận được tài nguyên có trọng số cao hơn của bất kì tài nguyên nào mà nó đang giữ. Muốn có tài nguyên j thì tiến trình phải giải phóng tất cả tài nguyên có trọng số i>j(nếu có).

2.1.5 Tránh Deadlock

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.

Một số khái niệm cơ sở

Trạng thái an toàn : trạng thái A là an toàn nếu hệ thống có thể thỏa mãn các nhu cầu tài nguyên (cho đến tối đa) của mỗi tiến trình theo một thứ tự nào đó mà vẫn ngăn chặn được tắc nghẽn.

Một chuỗi cấp phát an toàn: một thứ tự của các tiến trình <P1, P2,...,Pn> là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình Pi nhu cầu tài nguyên của Pi có thể được thỏa mãn với các tài nguyên còn tự do của hệ thống, cộng với các tài nguyên đang bị chiếm giữ bởi các tiến trình Pj khác, với j<i.

Một trạng thái an toàn không thể là trạng thái tắc nghẽn. Ngược lại một trạng thái không an toàn có thể dẫn đến tình trạng tắc nghẽn.

Chiến lược cấp phát : chỉ thỏa mãn yêu cầu tài nguyên của tiến trình khi trạng thái kết quả là an toàn!

2.2 Thuật giải 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.

Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker. Những cấu trúc dữ liệu này mã hoá trạng thái của hệ thống cấp phát tài nguyên. Gọi n là số quá 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 dùng của mỗi loại. Nếu Available[j]= k, có k thể hiện của loại tài nguyên Rj sẵn dùng.
- Max: một ma trận n x m định nghĩa số lượng tối đa yêu cầu của mỗi quá trình. Nếu

Max[i , j] = k, thì quá trình Pi có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên Rj.

- Allocation: một ma trận n x m định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp tới mỗi quá trình. Nếu Allocation[i, j] = k, thì quá trình Pi hiện được cấp k thể hiện của loại tài nguyên Rj.
- Need: một ma trận n x m hiển thị yêu cầu tài nguyên còn lại của mỗi quá trình. Nếu Need[i, j] = k, thì quá trình Pi có thể cần thêm k thể hiện của loại tài nguyên Rj để hoàn thành tác vụ của nó. Chú ý rằng, 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

Để đơn giản việc trình bày của giải thuật Banker, chúng ta thiết lập vài ký hiệu. Gọi X và Y là các vector có chiều dài n. Chúng ta nói rằng X £ Y nếu và chỉ nếu X[i] £ Y[i]cho tất cả i = 1, 2, …, n. Thí dụ, nếu X = (1, 7, 3, 2) và Y = (0, 3, 2, 1) thì Y £ X, Y<X nếu Y £ X và Y ¹ X.

2.3 Xây dựng chương trình mô phỏng thuật toán Banker

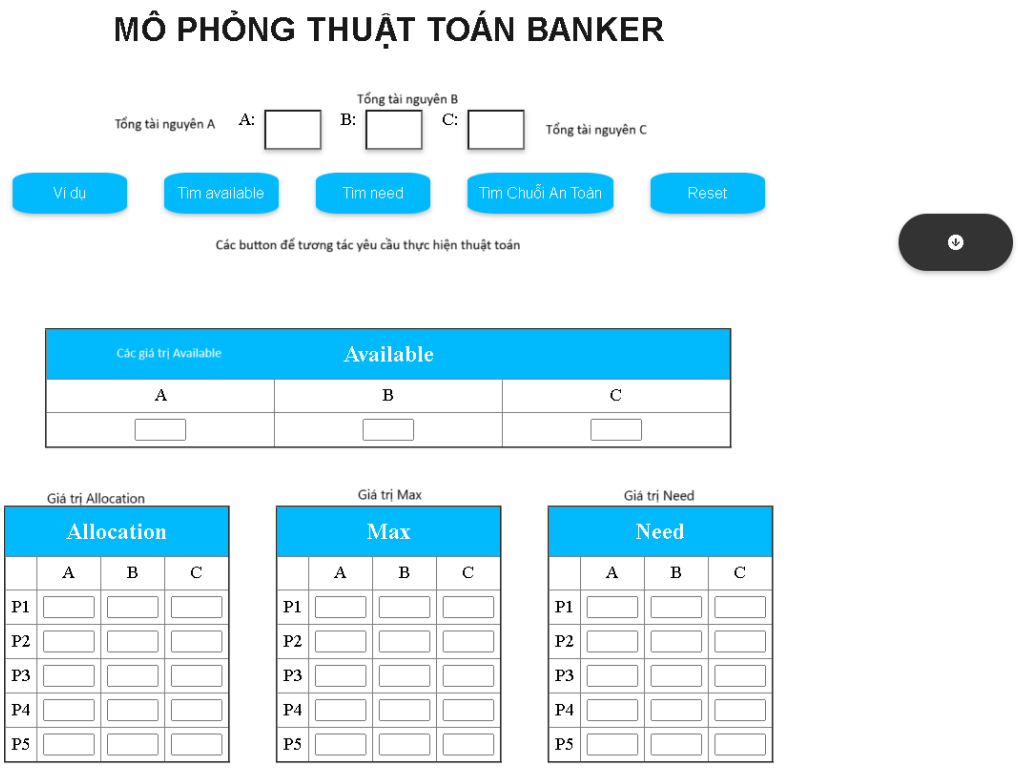
BANKER ALGORITHM

2.3.1 Chương trình giải thuật Banker

(https://github.com/NhanHcmue/Final_HDH)

2.3.2 Giao diện người dùng

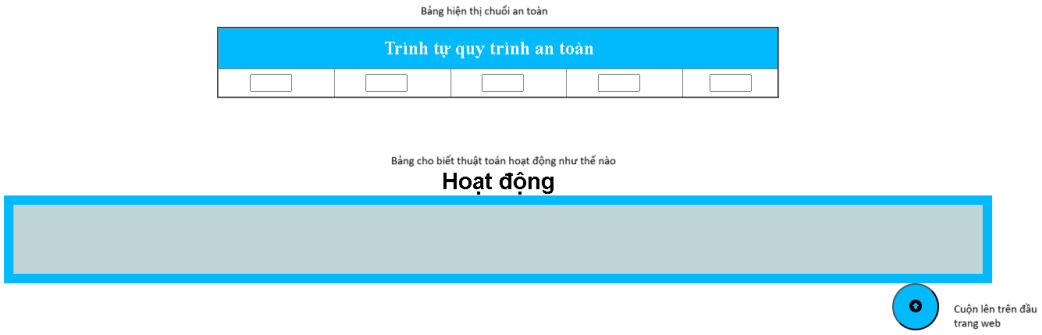
Chúng tôi sử dụng Html và css để code giao diện người dùng, dưới đây là hình ảnh demo web của chúng tôi



Hình 1: Giao diện người dùng

Phía trên gồm có 3 ô text dùng để nhập số tài nguyên của A, B, C sau đó đi xuống có năm nút gồm 5 chức năng khác nhau. Nút đầu tiên là vi dụ nó dùng để thêm các số chúng tôi đã cho sẵn, giúp mọi người dùng thử trang web một cách nhanh chống, nút thứ hai tìm available để thực hiện tính toán qua các giá trị có trong các

ô để tìm ra giá trị available, tìm need, chuỗi an toàn cũng tương tự chức năng trên, nút cuối cùng là reset dùng để đưa các giá trị về 0. Phía dưới là bảng Available dùng để hiện thị



Hình 2: Giao diện hiển thị hoạt động của thuật toán

2.3.3 Các hàm trong chương trình

Line 13 – 38 (Hàm validation)

```
function validation() {
  var a = document.getElementById("resourceA").value;
  var b = document.getElementById("resourceB").value;
  var c = document.getElementById("resourceC").value;
  if (a == " " || a < 0 || b == " " || b < 0 || c == " " || c < 0) {
    alert("Resource instance value can't be negative or blank");
  }
  for (var i = 1; i <= 5; i++) {
    for (var j = 1; j <= 3; j++) {
      if (
        document.getElementById("a" + i + j).value < 0 ||
        document.getElementById("a" + i + j).value == " "
      ) {
        alert("Allocation matrix elements can't be negative or blank");
      }
      if (
        document.getElementById("m" + i + j).value < 0 ||
        document.getElementById("m" + i + j).value == ""
      ) {
        alert("Max matrix elements can't be negative or blank");
      }
    }
    document.getElementById("calc" + i).value = "";
  }
  document.getElementById("calc0").value = "";
}
```

Hình 3: Hàm validation

Hàm validation() được dùng để kiểm tra tính hợp lệ của các giá trị khi nhập vào, đảm bảo rằng không có giá trị nào âm hoặc để trống. Nếu phát hiện thấy giá trị nào không hợp lệ, hàm sẽ hiển thị thông báo cho người dùng.

- Line 40 – 57 (Hàm reset)

```
function reset() {
  document.getElementById("resourceA").value = "";
  document.getElementById("resourceB").value = "";
  document.getElementById("resourceC").value = "";
  for (var i = 1; i <= 5; i++) {
    for (var j = 1; j <= 3; j++) {
      document.getElementById("a" + i + j).value = "";
      document.getElementById("m" + i + j).value = "";
      document.getElementById("n" + i + j).value = "";
    }
    document.getElementById("p" + i).value = "";
    document.getElementById("calc" + i).innerHTML = "";
  }
  document.getElementById("av11").value = "";
  document.getElementById("av12").value = "";
  document.getElementById("av13").value = "";
  document.getElementById("calc0").innerHTML = "";
}
```

Hình 4: Hàm reset

Hàm reset() xóa toàn bộ giá trị resource, Allocation matrix, Max matrix, Need matrix, làm mới tất cả các trường nhập liệu và kết quả, người dùng bắt đầu nhập lại từ đầu.

- Line 59 – 85 (Hàm sample)

```
function sample() {
  sam = [
    [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],
  ];
  for (var i = 1; i <= 5; i++) {
    for (var j = 1; j <= 3; j++) {
      document.getElementById("a" + i + j).value = sam[i - 1][j - 1];
      document.getElementById("m" + i + j).value = max[i - 1][j - 1];
    }
  }
  document.getElementById("resourceA").value = 10;
  document.getElementById("resourceB").value = 5;
  document.getElementById("resourceC").value = 7;
  document.getElementById("calc0").innerHTML = " Sample Loaded...";
}
```

Hình 5: Hàm sample

Hàm sample() được dùng để tự độn điền dữ liệu mẫu vào các trường nhập liệu. Nó thiết lập giá trị cho sam matrix, Max matrix, resource. Sau khi điền dữ liệu thì hiển thị thông báo Sample Loaded cho người dùng.

- Line 87 – 169 (Hàm find_avai)

```
function find_avai() { //find available
  var a = document.getElementById("resourceA").value;
  var b = document.getElementById("resourceB").value;
  var c = document.getElementById("resourceC").value;
  var x = 0;
  var y = 0;
  var z = 0;
  document.getElementById("calc1").innerHTML =
    "Available[n] = Tổng số tài nguyên - (Allocation[0][n] + Allocation[1][n] + Allocation[2][n] + Allocation[3][n] + Allocation[4
    "<br/>";
  document.getElementById("calc1").innerHTML +=
    "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
    "Available[0] = " +
    a +
    " - ";
  document.getElementById("calc2").innerHTML =
    "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
    "Available[1] = " +
    b +
    " - ";
  document.getElementById("calc3").innerHTML =
    "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
    "Available[2] = " +
    c +
    " - ";
  for (var i = 1; i <= 5; i++) {
    var x = x + parseInt(document.getElementById("a" + i + "1").value);
    document.getElementById("calc1").innerHTML +=
      "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
```

Hình 6: Hàm find_avai

Hàm find_avai() tính toán giá trị Available và hiển thị số lượng tài nguyên bằng cách lấy tổng số tài nguyên trừ đi số tài nguyên đã được phân bố cho mỗi Allocation, cập nhật các trường giá trị và hiển thị chi tiết quá trình tính toán.

- Line 171 – 201 (Hàm find_need)

```
function find_need() { //find need
  document.getElementById("calc1").innerHTML =
    "Need[n][n] = Max[n][n] - Allocation[n][n]" +
    "<br/>";
  document.getElementById("calc2").innerHTML = "";
  document.getElementById("calc3").innerHTML = "";
  document.getElementById("calc4").innerHTML = "";
  document.getElementById("calc5").innerHTML = "";
  document.getElementById("calc0").innerHTML = "";
  for (var i = 1; i <= 5; i++) {
    for (var j = 1; j <= 3; j++) {
      document.getElementById("n" + i + j).value =
        parseInt(document.getElementById("m" + i + j).value) -
        parseInt(document.getElementById("a" + i + j).value);
      document.getElementById("calc" + i).innerHTML +=
        "<span style='font-size: 16px; color: #800000; font-family: Arial;'>Need[" +
        (i - 1) +
        "]" +
        "[" +
        (j - 1) +
        "]" +
        " = " +
        document.getElementById("m" + i + j).value +
        " - " +
        document.getElementById("a" + i + j).value +
        " = " +
        document.getElementById("n" + i + j).value +
        "      </span>";
    }
  }
```

Hình 7: Hàm need

Hàm find_need() tính toán và hiên thị Need matrix bằng cách lấy Max matrix trừ cho Allocation matrix, hàm này cũng cập nhật các trường giá trị và hiện thị chi tiết quá tình tính toán.

- Line 203 – 297 (Hàm find_sequence)

Hàm find_sequence() tìm kiếm chuỗi an toàn trong hệ thống bằng cách sử dụng thuật toán Banker. Hàm này tính toán và hiển thị quá trình kiểm tra từng bước để xác định liệu hệ thống có thể tiến đến trạng thái an toàn hay không. Nếu không tìm thấy chuỗi an toàn, hệ thống sẽ hiển thị thông báo “Deadlock!” và đặt lại các giá trị.

```
function find_sequence() { //find safe
  var dp = 0;
  var checker = 0;
  var q = 1;
  var k = 1;

  for (var j = 1; j <= 5; j++) {
    x1 = parseInt(document.getElementById("av11").value); //gán giá trị available
    x2 = parseInt(document.getElementById("av12").value);
    x3 = parseInt(document.getElementById("av13").value);
    document.getElementById("calc" + j).innerHTML =
      "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
      "Step" + j + ": "
    for (var i = k; i <= 5; i++) {
      var ex1 = parseInt(document.getElementById("a" + i + "1").value); //gán giá trị Allocation
      var ex2 = parseInt(document.getElementById("a" + i + "2").value);
      var ex3 = parseInt(document.getElementById("a" + i + "3").value);
      if (ex1 != 0 || ex2 != 0 || ex3 != 0) {
        if (
          x1 >= parseInt(document.getElementById("n" + i + "1").value) &&
          x2 >= parseInt(document.getElementById("n" + i + "2").value) &&
          x3 >= parseInt(document.getElementById("n" + i + "3").value)
        ) {
          document.getElementById("p" + q).value = "P" + i;
          document.getElementById("calc" + j).innerHTML +=
            "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
            "   Need[" +
            i +
            "] = (" +
```

Hình 8: Hàm find_sequence 1

```
            "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
            "   Need[" +
            i +
            "] = (" +
            document.getElementById("n" + i + "1").value +
            ", " +
            document.getElementById("n" + i + "2").value +
            ", " +
            document.getElementById("n" + i + "3").value +
            ") < Available = (" +
            x1 +
            ", " +
            x2 +
            ", " +
            x3 +
            ") => Process P" +
            i +
            " được chọn, Available mới là (" +
            document.getElementById("av11").value +
            ", " +
            "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
            document.getElementById("av12").value +
            ", " +
            document.getElementById("av13").value +
            ") + (" +
            document.getElementById("a" + i + "1").value +
            ", " +
            document.getElementById("a" + i + "2").value +
            ", " +
```

Hình 9: Hàm find_sequence 2

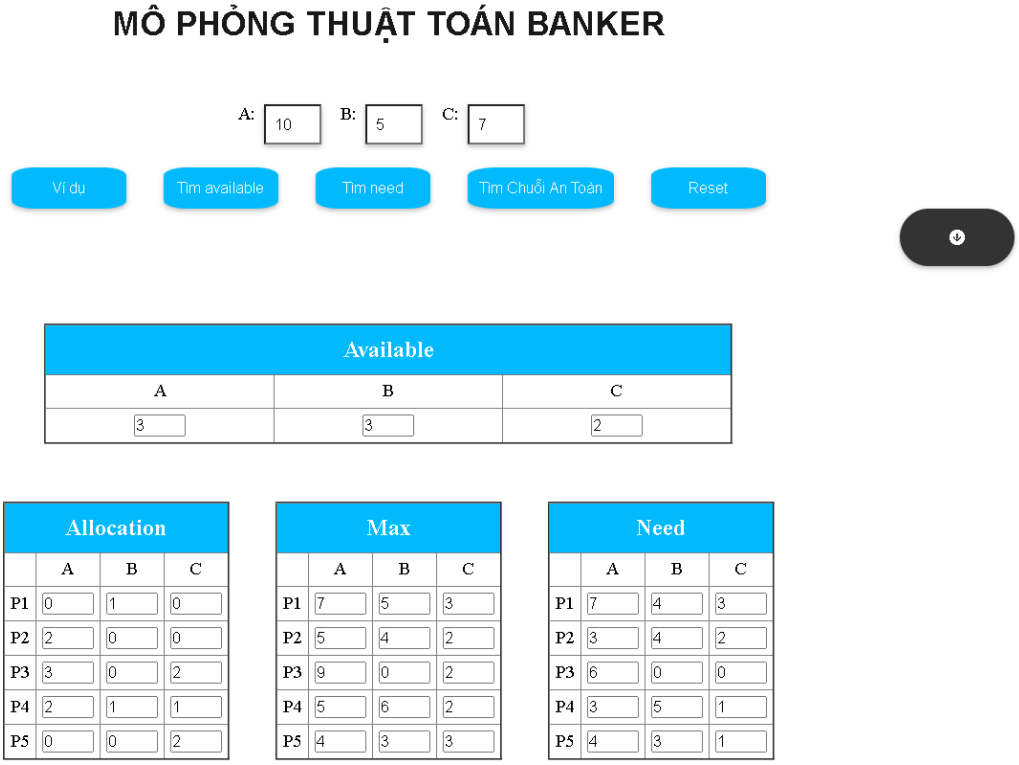
```
            document.getElementById("calc" + j).innerHTML +=
              "<span style='font-size: 16px; color: #800000; font-family: Arial;'>" +
              "(" +
              document.getElementById("av11").value +
              ", " +
              document.getElementById("av12").value +
              ", " +
              document.getElementById("av13").value +
              ")";
            document.getElementById("a" + i + "1").value = "0";
            document.getElementById("a" + i + "2").value = "0";
            document.getElementById("a" + i + "3").value = "0";
            k = i + 1;
            if (k == 6) {
              k = 1;
            }
            q = q + 1;
            checker = checker + 1;
            break;
          } else {
            dp = i;
          }
        }
      }
    }
    if (checker == 0) {
      alert("Deadlock!");
      reset();
    }
}
```

Hình 10: Hàm find_sequence 3

2.3.4 Dữ liệu mẫu

- Trường hợp Deadlock

Input



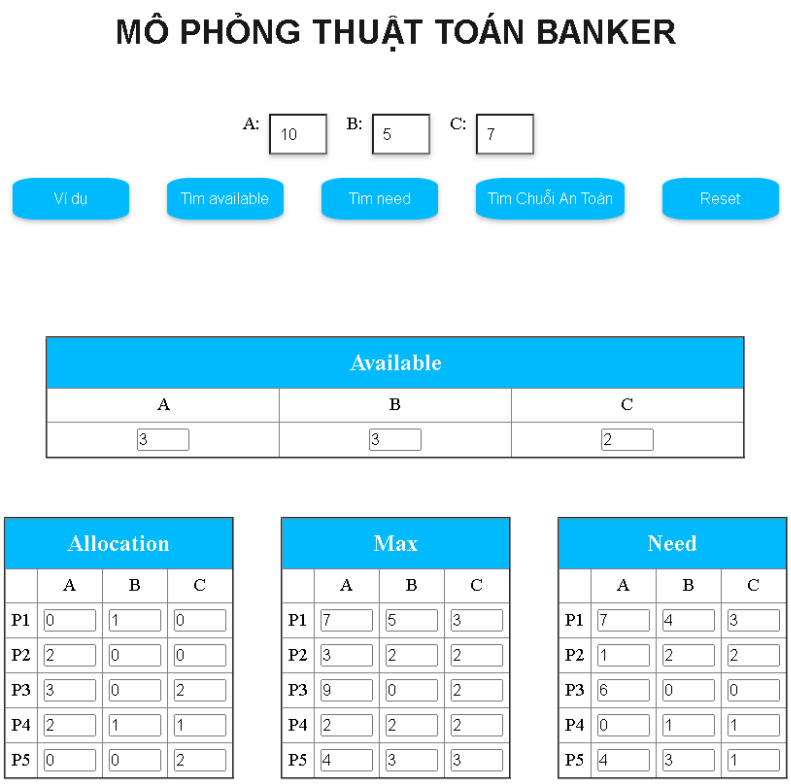
Hình 11: Ví dụ mẫu deadlock

Output

Hình 12: Ví dụ mẫu deadlock

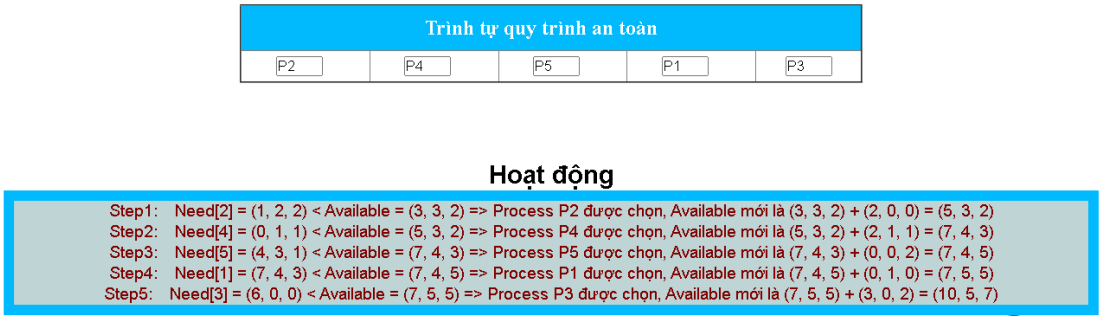
- Trường hợp an toàn

Input



Hình 13: Ví dụ về trường hợp an toàn

Output



Hình 14: Ví dụ về trường hợp an toàn

Tài liệu tham khảo

[1] G. v. .. T. H. Nhi, Giáo trình hệ điều hành, Giảng viên . Lê Khắc Nhiên Ân.

[2] V. H. T. T. Long, “Tiểu luận hệ điệu hành,” Đại học sư phạm, Hồ Chí Minh, 2022.