

NGUYÊN LÝ HỆ ĐIỀU HÀNH

Giảng viên: TS. Đoàn Thị Quế
Bộ môn Mạng và an toàn thông tin

Chương 3

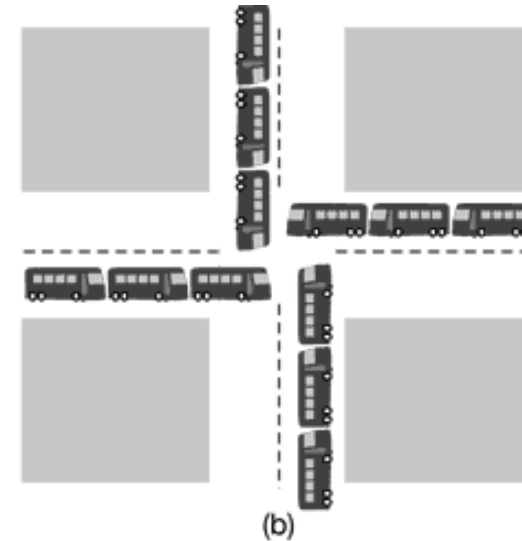
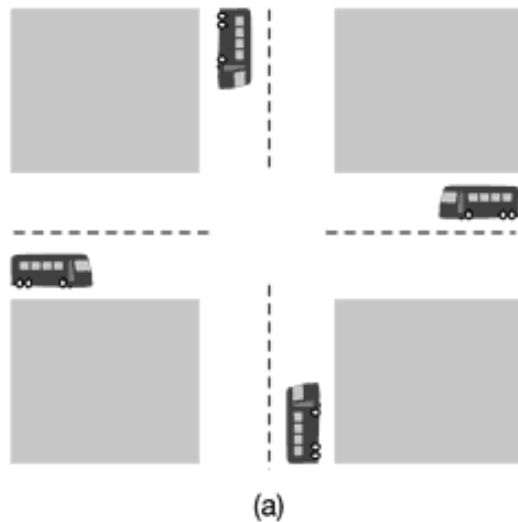


SỰ BẾ TẮC (DEADLOCK)

- Sự bế tắc
- Các khái niệm về bế tắc
- Xử lý bế tắc

3.1 – Sự bế tắc

- Khi có hai hoặc nhiều tiến trình tác động lẫn nhau, chúng có thể gây ra xung đột và không giải quyết được. Hiện tượng đó được gọi là sự bế tắc (**deadlock**).
- Ví dụ sự bế tắc xảy ra trong giao thông



Sự bế tắc trong máy tính

- Trong hệ thống máy tính có những loại tài nguyên chỉ cho phép một tiến trình được sử dụng ở một thời điểm
 - Ví dụ: Máy in, ổ băng từ, ...
 - Việc hai tiến trình đồng thời sử dụng tài nguyên này sẽ gây ra bế tắc.
- Hệ điều hành phải có tính năng cho phép giải quyết bế tắc

Ví dụ: Xét tình huống sau đây

- Hai tiến trình A, B cùng muốn scan ảnh rồi ghi file ảnh vào đĩa CD.
- Tiến trình A gửi yêu cầu muốn được cấp quyền sử dụng scanner và máy ghi CD
- Tiến trình B gửi yêu cầu muốn được cấp quyền sử dụng máy ghi CD và scanner

(lúc đó cả máy CD và scanner đều đang rồi)

- Tiến trình A được cấp quyền sử dụng scanner
- Tiến trình B được cấp quyền sử dụng máy CD

*Cả hai tiến trình cùng phải chờ để được cấp nốt tài nguyên còn lại, quá trình chờ đợi là mãi mãi ⇒ **Bế tắc xảy ra***

3.2 – Các khái niệm về bế tắc

- Định nghĩa bế tắc
- Bốn điều kiện xảy ra bế tắc
- Mô hình hoá sự bế tắc

1. Định nghĩa bế tắc

- *“Một tập hợp các tiến trình bị coi là bế tắc nếu mỗi tiến trình trong tập hợp phải chờ một sự kiện, mà sự kiện đó lại chỉ có thể do một tiến trình khác trong tập hợp tạo ra.”*
- Tất cả các tiến trình đều đang phải chờ, không tiến trình nào trong số chúng có thể tạo ra sự kiện cần thiết để đánh thức các thành viên trong tập hợp, nên chúng sẽ phải chờ mãi mãi.

2. Bốn điều kiện xảy ra bế tắc

Coffman và các cộng sự (1971) đã chỉ ra bốn điều kiện xảy ra bế tắc sau đây:

- 1) Loại trừ tương hỗ:** Mỗi tài nguyên hoặc được sở hữu bởi một tiến trình duy nhất, hoặc đang rảnh rỗi. (1 tài nguyên chỉ phục vụ 1 tiến trình)
- 2) Giữ và chờ:** Các tiến trình đang nắm giữ tài nguyên được cấp trước đó có thể gửi yêu cầu đòi cấp tài nguyên mới
- 3) Không giải phóng:** Tài nguyên do tiến trình giữ không thể phân phối lại cho tiến trình khác trừ khi tiến trình đang giữ tự nguyện giải phóng tài nguyên
- 4) Chờ đợi vòng tròn:** Phải có một hàng đợi vòng tròn gồm hai hoặc nhiều tiến trình, mỗi tiến trình lại đang chờ một tài nguyên được sở hữu bởi chính thành viên tiếp theo trong hàng đợi

Cả bốn điều kiện này phải có mặt khi xảy ra bế tắc. Nếu vắng mặt một trong số đó thì sẽ không có bế tắc..

3. Mô hình hoá sự bế tắc

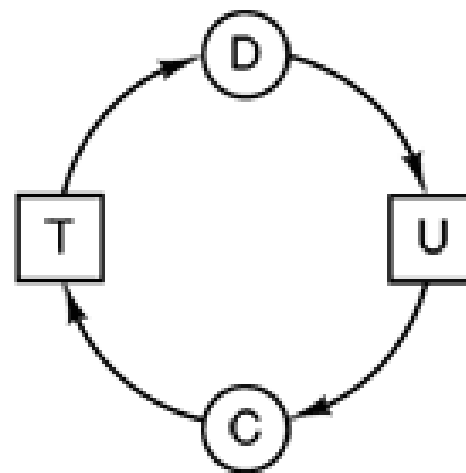
- Holt đã đưa ra cách mô hình hoá bốn điều kiện trên bằng các sơ đồ nút (năm 1972)



(a)



(b)



(c)

Sơ đồ phân phối tài nguyên: (a) Giữ tài nguyên; (b) Yêu cầu tài nguyên; (c) Bế tắc xảy ra

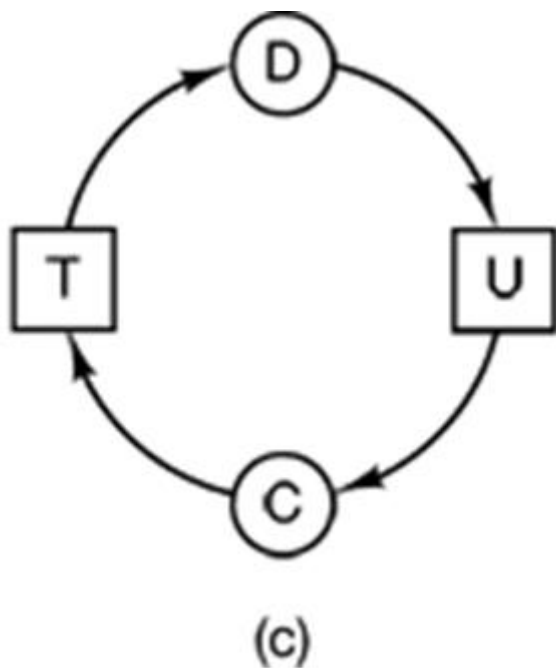


(a)



(b)

- Nút tiến trình được khoanh tròn
- Nút tài nguyên được đóng khung vuông
- Mũi tên được nối từ nút tài nguyên tới nút tiến trình nghĩa là tài nguyên đó do tiến trình đó yêu cầu, và đã được phân cho tiến trình, tiến trình hiện đang sở hữu nó (*hình a*)
- Mũi tên được nối từ tiến trình tới tài nguyên, nghĩa là tiến trình đó hiện đang bị dừng để chờ nhận được tài nguyên đó (*hình b*)



Xét hình c:

- Tiến trình C đang chờ tài nguyên T
- Tài nguyên T lại do tiến trình D nắm giữ
- Tiến trình D không thể giải phóng T vì nó đang chờ tài nguyên U
- Tài nguyên U lại do C nắm giữ và không thể được giải phóng do C đang chờ tài nguyên T

➡ ***Tạo thành một vòng khép kín, bế tắc xảy ra.***

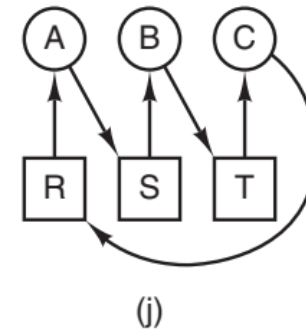
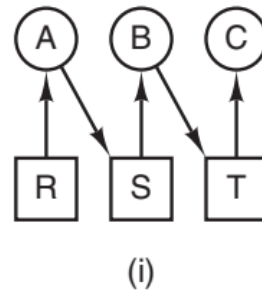
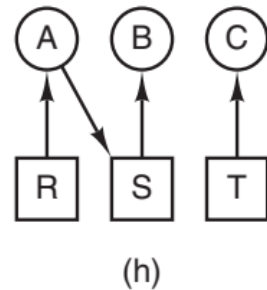
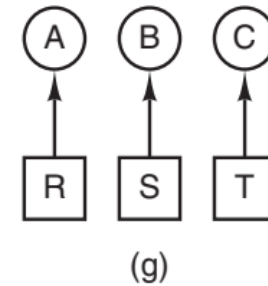
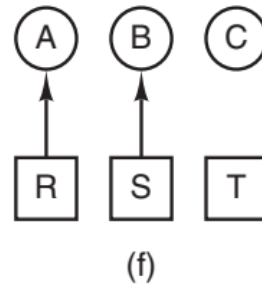
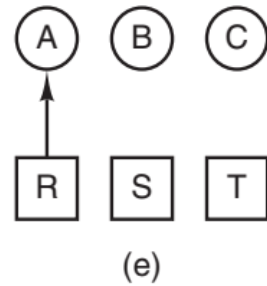
Ví dụ: bế tắc

A
Request R
Request S
Release R
Release S
(a)

B
Request S
Request T
Release S
Release T
(b)

C
Request T
Request R
Release T
Release R
(c)

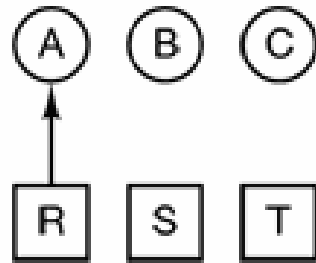
1. A requests R
 2. B requests S
 3. C requests T
 4. A requests S
 5. B requests T
 6. C requests R
- deadlock



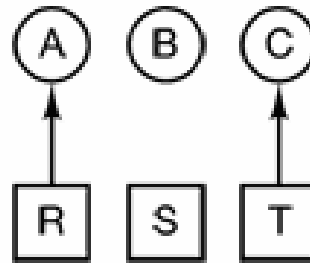
Cách phòng tránh: *Treo tiến trình B (hệ điều hành chỉ cho chạy A và C)*

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
no deadlock

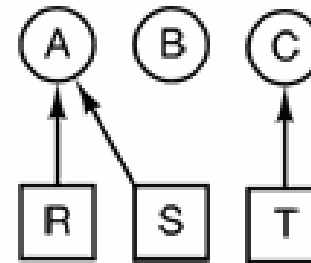
(k)



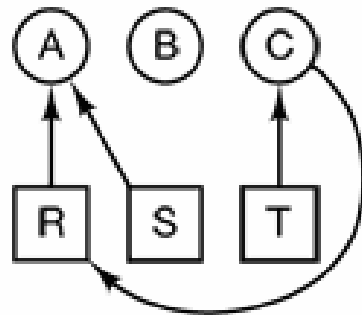
(l)



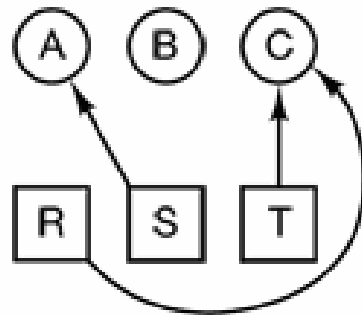
(m)



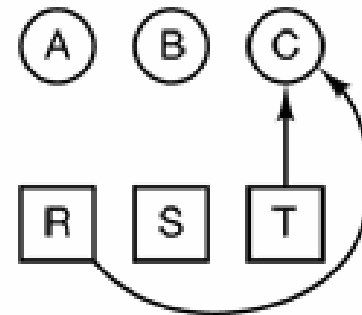
(n)



(o)



(p)



(q)

- Sau bước (q) có thể tiếp tục chạy tiến trình B và cấp cho nó tài nguyên S mà không gây ra bế tắc

→ *Nếu việc thực hiện một yêu cầu của tiến trình có nguy cơ dẫn đến bế tắc, hệ điều hành có thể treo tiến trình mà không cần thực hiện yêu cầu đó cho tới khi thấy an toàn!*

3.3 – Xử lý bế tắc

Bốn chiến lược xử lý bế tắc

- 1) Bỏ qua tất cả các vấn đề (Giải thuật đả điều)
- 2) Để cho các bế tắc xảy ra, phát hiện chúng và xử lý (*)
- 3) Chủ động phòng tránh bằng cách phân phối tài nguyên thật cẩn thận
- 4) Ngăn chặn, bằng cách loại bỏ sự tồn tại của một trong bốn điều kiện cần thiết gây ra bế tắc

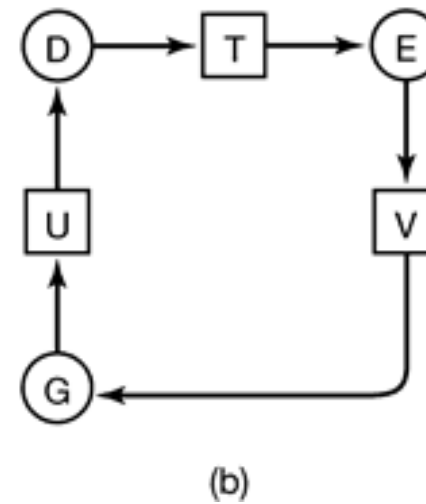
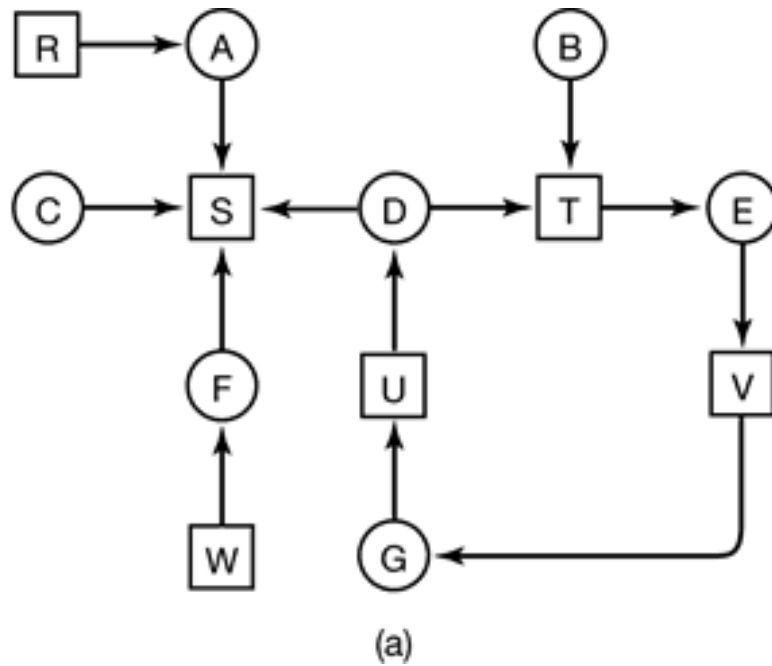
(*) Ta sẽ tập trung chủ yếu vào vấn đề này

Phát hiện bế tắc

- Xét trường hợp số lượng tài nguyên thuộc mỗi loại bằng một
- **Các phương pháp phát hiện bế tắc:**
 - Vẽ sơ đồ phân phối tài nguyên
 - ◆ Nếu sơ đồ có một vòng tròn khép kín bế tắc thì bế tắc xảy ra
 - Giải thuật phát hiện vòng kín bế tắc
- **Ví dụ:** Xét hệ thống gồm có bảy tiến trình (từ A đến G), và có sáu tài nguyên (từ R đến W). Trạng thái của các tài nguyên và tiến trình như sau:
 - Tiến trình A đang nắm tài nguyên R và muốn có thêm S.
 - Tiến trình B không nắm tài nguyên nào và đang muốn có T.
 - Tiến trình C không nắm tài nguyên nào và đang muốn có S.
 - Tiến trình D đang sở hữu U và muốn có S và T.
 - Tiến trình E đang sở hữu T và muốn có V
 - Tiến trình F sở hữu W và muốn có S.
 - Tiến trình G sở hữu V và muốn có U.

“Hệ thống này có bị bế tắc không? và nếu có thì sẽ bao gồm những tiến trình nào?” ¹⁶

Vẽ sơ đồ phân phối tài nguyên



Kết luận:

- Các tiến trình D, E, G bị bế tắc vì tạo thành vòng kín
- Các tiến trình A, C, và F không bị bế tắc vì S có thể được phân phối cho chúng, khi dùng xong chúng có thể trả lại để tiến trình khác sử dụng

Bài tập

- 4 tiến trình: A, B, C, D; 4 tài nguyên: T, X, Y, Z
 - A sở hữu X và xin cấp Y
 - B xin cấp X và T
 - C sở hữu Y, xin cấp Z và T
 - D sở hữu Z, xin cấp Y

Hệ thống có bị bế tắc không? Nếu có thì những tiến trình nào bị bế tắc?

Giải thuật phát hiện vòng kín

- Có rất nhiều giải thuật khác nhau để tự động phát hiện ra vòng kín, dưới đây ta xem xét một giải thuật đơn giản

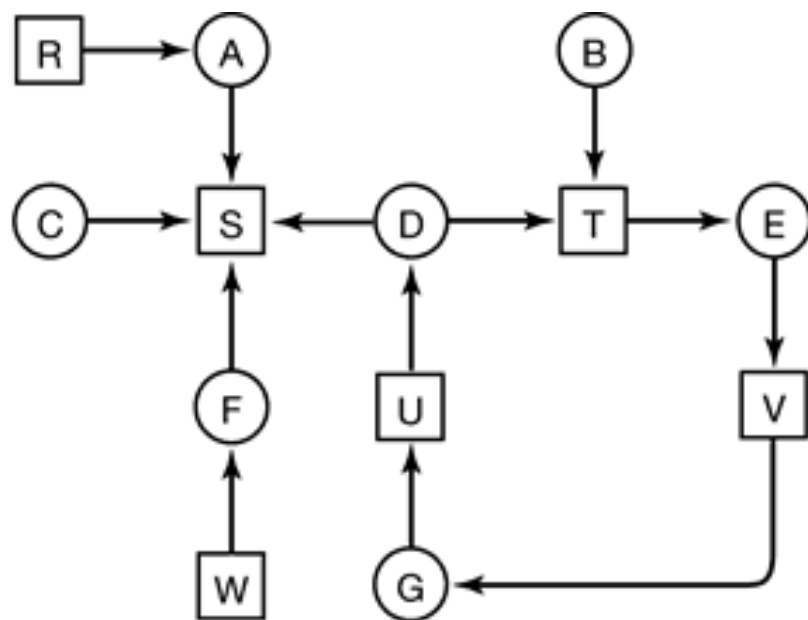
Tìm hiểu các giải thuật phát hiện vòng kín khác trong tài liệu *EVEN, S.: Graph Algorithms, Potomac, MD: Computer Science Press, 1979*

Các bước của giải thuật phát hiện vòng kín đơn giản

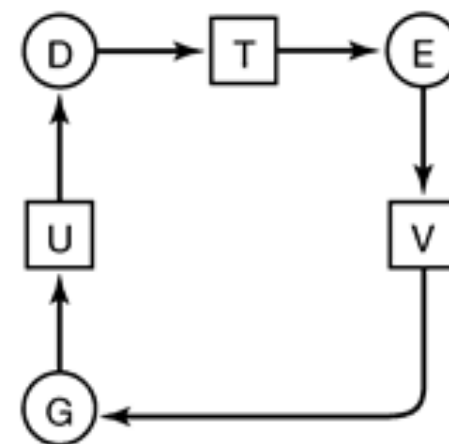
Giả sử có N nút trong sơ đồ. *Lần lượt thực hiện 5 bước sau đây cho từng nút.*

1. Khởi tạo L (là một danh sách rỗng) và đảm bảo rằng tất cả các mũi tên đều chưa bị đánh dấu.
2. Đặt nút hiện hành vào cuối danh sách L và kiểm tra xem liệu nút này có xuất hiện trong danh sách hai lần không: Nếu có thì tức là đã tạo thành vòng kín, giải thuật kết thúc; nếu không thì sang bước 3.
3. Kiểm tra nút hiện hành xem có mũi tên nào đi ra mà chưa được đánh dấu không. Nếu có thì chuyển sang bước 4, nếu không có thì chuyển sang bước 5.
4. Chọn ngẫu nhiên một mũi tên (trong số các mũi tên chưa được đánh dấu) rồi tiến hành đánh dấu nó. Đi theo hướng mũi tên đó để đến một nút mới, rồi quay trở lại bước 2.
5. Ta đã đi đến điểm cuối. Xóa bỏ nó và quay trở lại nút trước đó, tiếp tục áp dụng bước 2. Nếu nút này chính là nút ban đầu, tức là không phát hiện ra vòng kín nào theo nhánh này, hoàn thành quá trình kiểm tra với nút hiện hành.

Ví dụ minh họa



(a)

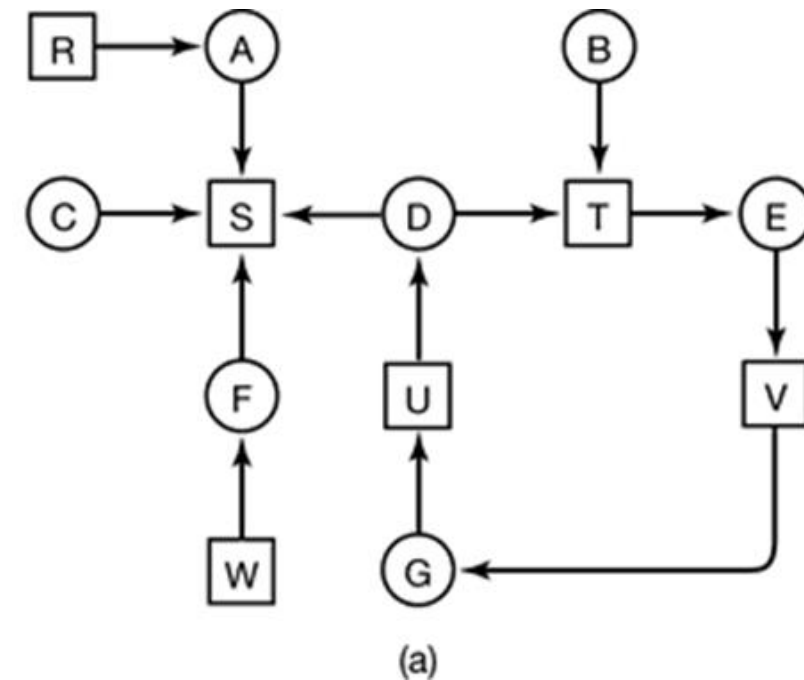


(b)

- Trật tự xử lý các nút là bất kỳ.
- Giả sử lần lượt kiểm tra các nút theo thứ tự: $R, A, B, C, S, D, T, E, F...$ Nếu phát hiện ra một vòng kín thì giải thuật kết thúc.

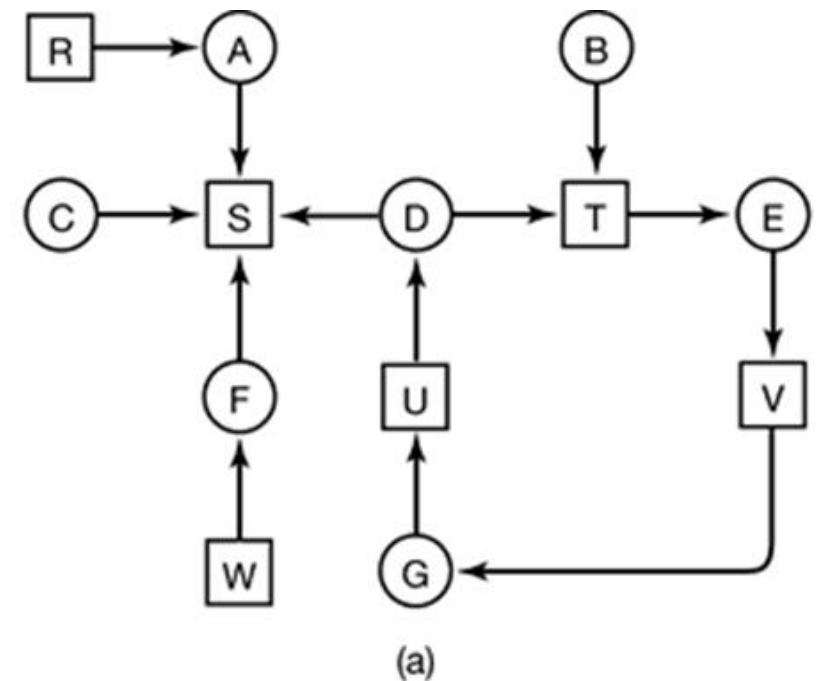
Bắt đầu từ nút R:

- Ban đầu L là một danh sách rỗng
- Đặt R vào danh sách, $L = [R]$. Đi tới nút tiếp theo là A, thêm A vào danh sách, $L = [R, A]$.
- Từ A ta đi tới S, thêm S vào danh sách, $L = [R, A, S]$.
- S không có mũi tên đi ra nào, nó là một điểm cuối, nên ta sẽ quay trở lại A, $L = [R, A]$. Do A không có mũi tên đi ra nào chưa được đánh dấu, nên ta sẽ quay lại R, $L = [R]$, hoàn thành việc kiểm tra với nút R.



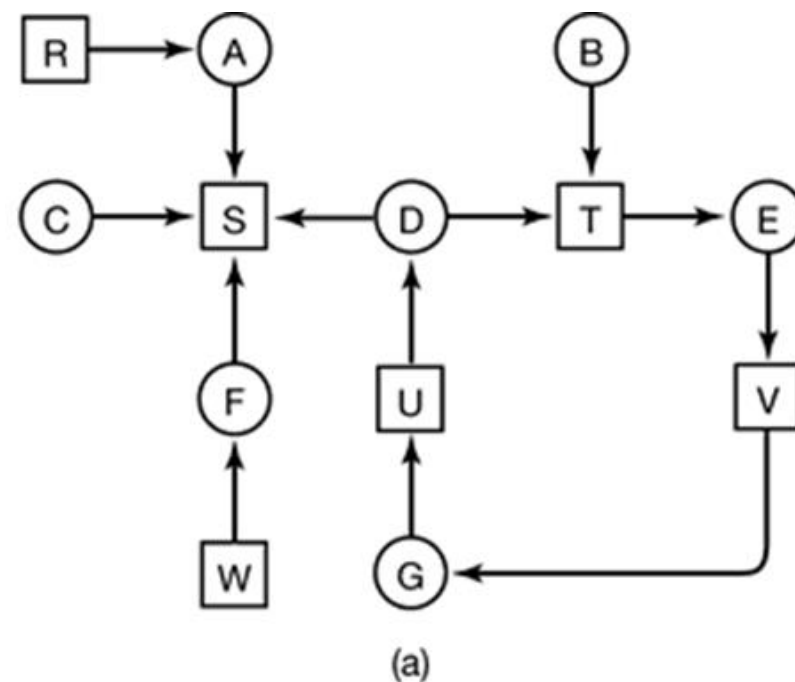
Tiếp theo sẽ bắt đầu từ A:

- Danh sách L được làm rỗng
- Đặt A vào danh sách $L = [A]$, rồi đi tới nút tiếp theo là S, thêm S vào danh sách $L = [A, S]$.
- S là một điểm cuối, nên ta sẽ quay trở lại A, $L = [A]$, hoàn thành việc kiểm tra với nút A.



Tiếp theo sẽ xuất phát từ B:

- Từ B ta đi theo các mũi tên hướng ra cho tới khi gặp D, lúc đó $L = [B, T, E, V, G, U, D]$.
- Tại D ta sẽ phải lựa chọn ngẫu nhiên nút tiếp theo.
 - ◆ Nếu chọn nút S, đó là một điểm cuối và ta sẽ quay lại D, lúc đó $L = [B, T, E, V, G, U, D]$. Từ D đi tới T, thêm T vào danh sách $L = [B, T, E, V, G, U, D, T]$ > phát hiện vòng lặp kín, giải thuật kết thúc.
 - ◆ Nếu chọn nút T, đặt T vào danh sách: $L = [B, T, E, V, G, U, D, T]$ > phát hiện vòng lặp kín, giải thuật kết thúc.



Giải quyết bế tắc

Sau khi phát hiện bế tắc cần làm gì để hệ thống hoạt động trở lại?

- 1) Biện pháp giải phóng tài nguyên
- 2) Biện pháp quay trở lại
- 3) Biện pháp huỷ bỏ tiến trình

Biện pháp giải phóng tài nguyên

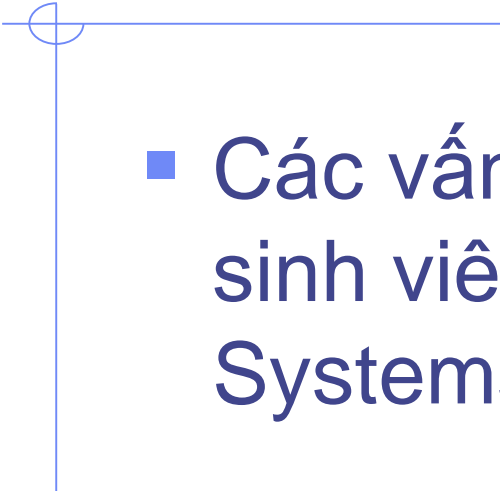
- Trong một số trường hợp, ta có thể tạm lấy tài nguyên của một tiến trình nào đó và giao cho tiến trình khác
- Biện pháp này phụ thuộc rất nhiều vào thuộc tính của tài nguyên đó và không dễ thực hiện.

Biện pháp quay trở lại

- Tiến trình đang sở hữu tài nguyên sẽ bị “reset” về thời điểm trước khi nó giành được tài nguyên. Tất cả các công việc đã làm sau thời điểm đó sẽ bị mất.
- Tài nguyên đó có thể được phân phối cho một tiến trình khác đang bế tắc.
- Khi tiến trình được khởi động trở lại, nếu nó muốn giành lấy tài nguyên, nó sẽ phải chờ cho tới khi tài nguyên đó sẵn sàng
- Cần có cơ chế lưu trữ trạng thái của tiến trình trong quá khứ, để có thể quay trở lại.

Biện pháp huỷ bỏ tiến trình

- Biện pháp thô bạo nhất, nhưng cũng đơn giản nhất để giải quyết bế tắc là huỷ bỏ một hoặc nhiều tiến trình.
- Có thể huỷ bỏ một tiến trình trong vòng kín.
 - Nếu may mắn thì các tiến trình còn lại có thể tiếp tục hoạt động.
 - Còn nếu chưa được thì có thể tiếp tục huỷ bỏ các tiến trình khác cho tới khi vòng kín bị phá vỡ.

- 
- Các vấn đề *Phòng tránh bế tắc, Ngăn chặn bế tắc...* sinh viên tự nghiên cứu trong cuốn Modern Operating Systems.



Hết Chương 3