

Chapter 2.

Process Management

- Processes
- Threads
- CPU Scheduling
- Process Synchronization
- Deadlocks

Tiến trình



Mô hình tiến
trình

Các trạng thái
của tiến trình

Chế độ xử lý
của tiến trình

Cấu trúc dữ
liệu khôi quản
lý tiến trình

Các thao tác
trên tiến trình

Cấp phát tài
nguyên cho
tiến trình

Mô hình tiến trình

- Tiến trình là một chương trình đang xử lý, sở hữu một con trỏ lệnh, tập các thanh ghi và các biến.
- Mỗi tiến trình có duy nhất một không gian địa chỉ và chỉ có một dòng xử lý.
- Để hoàn thành tác vụ của mình, một tiến trình có thể cần đến một số tài nguyên như CPU, bộ nhớ chính, các tập tin, các thiết bị nhập xuất.
- Phân biệt khái niệm chương trình và khái niệm tiến trình ?
 - Một chương trình là một thực thể thụ động, chứa đựng các chỉ thị điều khiển máy tính để tiến hành một tác vụ nào đó;
 - Khi cho thực hiện các chỉ thị này, chương trình chuyển thành tiến trình, là một thực thể hoạt động.

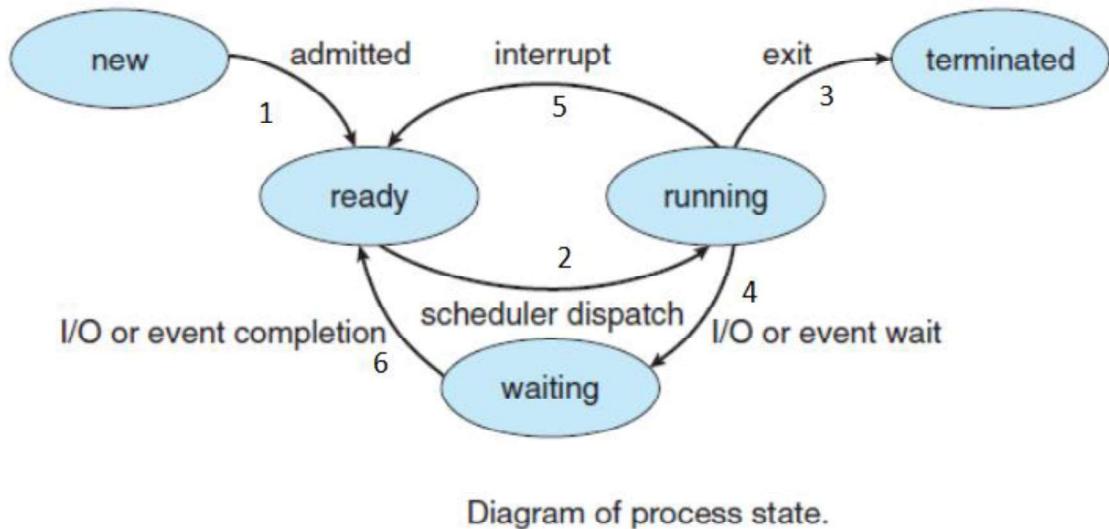
- Các tiến trình hoàn toàn độc lập với nhau, chỉ có thể liên lạc thông qua các cơ chế thông tin giữa các tiến trình mà hệ điều hành cung cấp.
- **Sự đa chương:**
 - Các nhà thiết kế hệ điều hành đề xuất một mô hình song song giả lặp bằng cách chuyển đổi bộ xử lý qua lại giữa các chương trình để duy trì hoạt động của nhiều chương trình cùng lúc.
 - Sự chuyển đổi nhanh chóng này được gọi là sự đa chương.
- **Bộ điều phối:**
 - Hệ điều hành chịu trách nhiệm sử dụng một thuật toán điều phối để quyết định thời điểm cần dừng hoạt động của tiến trình đang xử lý để phục vụ một tiến trình khác, và lựa chọn tiến trình tiếp theo sẽ được phục vụ.
 - Bộ phận thực hiện chức năng này của hệ điều hành được gọi là bộ điều phối.

Các trạng thái của tiến trình

Tại một thời điểm, một tiến trình có thể nhận trong một các trạng thái sau đây:

- New: Tiến trình đang được tạo lập
- Running: Các chỉ thị của tiến trình đang được xử lý
- Waiting (blocked): Tiến trình chờ được cấp phát một tài nguyên, hay chờ một sự kiện xảy ra.
- Ready: Tiến trình chờ được cấp phát CPU để xử lý
- Terminated: Tiến trình hoàn tất xử lý.

Sơ đồ chuyển trạng thái của các tiến trình



Ghi chú:

Tại mỗi thời điểm, chỉ có một tiến trình có thể nhận trạng thái running trên một bộ xử lý bất kỳ. Trong khi đó, nhiều tiến trình có thể ở trạng thái waiting hay ready.

- (1) Tiến trình mới tạo được đưa vào hệ thống.
- (2) Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU.
- (3) Tiến trình kết thúc.
- (4) Tiến trình yêu cầu một tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng để cấp phát tại thời điểm đó; hoặc tiến trình phải chờ một sự kiện hay thao tác nhập xuất.
- (5) Bộ điều phối chọn một tiến trình khác để cho xử lý.
- (6) Tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát; hay sự kiện hoặc thao tác nhập/xuất tiến trình đang đợi hoàn tất.

Chế độ xử lý của tiến trình

- Để đảm bảo hệ thống hoạt động đúng đắn, hệ điều hành cần phải được bảo vệ khỏi sự xâm phạm của các tiến trình. Bản thân các tiến trình và dữ liệu cũng cần được bảo vệ để tránh các ảnh hưởng sai lệch lẫn nhau.
- Một cách tiếp cận để giải quyết vấn đề là phân biệt hai chế độ xử lý cho các tiến trình:
 - Chế độ không đặc quyền;
 - Chế độ đặc quyền nhờ vào sự trợ giúp của cơ chế phần cứng.

- Tập lệnh của CPU được phân chia thành các lệnh đặc quyền và lệnh không đặc quyền. Cơ chế phần cứng chỉ cho phép các lệnh đặc quyền được thực hiện trong chế độ đặc quyền.
- Thông thường chỉ có hệ điều hành hoạt động trong chế độ đặc quyền, các tiến trình của người dùng hoạt động trong chế độ không đặc quyền, không thực hiện được các lệnh đặc quyền có nguy cơ ảnh hưởng đến hệ thống; như vậy hệ điều hành được bảo vệ.
- Khi một tiến trình người dùng gọi đến một lời gọi hệ thống, tiến trình của hệ điều hành xử lý lời gọi này sẽ hoạt động trong chế độ đặc quyền, sau khi hoàn tất thì trả quyền điều khiển về cho tiến trình người dùng trong chế độ không đặc quyền.

Cấu trúc dữ liệu khối quản lý tiến trình

Hệ điều hành quản lý các tiến trình trong hệ thống thông qua **Khối quản lý tiến trình** (process control block -PCB). PCB là một vùng nhớ lưu trữ các thông tin mô tả cho tiến trình, với các thành phần chủ yếu bao gồm:

- **Định danh của tiến trình (1):** Giúp phân biệt các tiến trình
- **Trạng thái tiến trình (2):** Xác định hoạt động hiện hành của tiến trình.
- **Ngữ cảnh của tiến trình (3):** Mô tả các tài nguyên tiến trình đang trong quá trình, hoặc để phục vụ cho hoạt động hiện tại, hoặc để làm cơ sở phục hồi hoạt động cho tiến trình, bao gồm các thông tin về:
 - *Trạng thái CPU,*
 - *Bộ xử lý,*
 - *Bộ nhớ chính,*
 - *Tài nguyên sử dụng,*
 - *Tài nguyên tạo lập.*

- **Thông tin giao tiếp (4):** Phản ánh các thông tin về quan hệ của tiến trình với các tiến trình khác trong hệ thống: *Tiến trình cha, tiến trình con, độ ưu tiên.*
- **Thông tin thống kê (5):** Đây là những thông tin thống kê về hoạt động của tiến trình, như thời gian đã sử dụng CPU, thời gian chờ. Các thông tin này có thể có ích cho công việc đánh giá tình hình hệ thống và dự đoán các tình huống tương lai.

Các thao tác trên tiến trình

Hệ điều hành cung cấp các thao tác sau trên một tiến trình:

- Tạo lập tiến trình
- Kết thúc tiến trình
- Tạm dừng tiến trình
- Tái kích hoạt tiến trình
- Thay đổi độ ưu tiên tiến trình

Tạo lập tiến trình

- Trong quá trình xử lý, một tiến trình có thể tạo lập nhiều tiến trình mới bằng cách sử dụng một lời gọi hệ thống tương ứng.
- Tiến trình gọi lời gọi hệ thống để tạo tiến trình mới sẽ được gọi là tiến trình *cha*, tiến trình được tạo gọi là tiến trình *con*.
- Mỗi tiến trình con đến lượt nó lại có thể tạo các tiến trình mới...quá trình này tiếp tục sẽ tạo ra một *cây tiến trình*.
- Các công việc hệ điều hành cần thực hiện khi tạo lập tiến trình bao gồm:
 - Định danh cho tiến trình mới phát sinh,
 - Đưa tiến trình vào danh sách quản lý của hệ thống,
 - Xác định độ ưu tiên cho tiến trình,
 - Tạo process control block - PCB cho tiến trình,
 - Cấp phát các tài nguyên ban đầu cho tiến trình.

Kết thúc tiến trình

- Một tiến trình kết thúc xử lý khi nó hoàn tất chỉ thị cuối cùng và sử dụng một lời gọi hệ thống để yêu cầu hệ điều hành hủy bỏ nó. Đôi khi một tiến trình có thể yêu cầu hệ điều hành kết thúc xử lý của một tiến trình khác.
- Khi một tiến trình kết thúc, hệ điều hành thực hiện các công việc:
 - Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình,
 - Hủy tiến trình khỏi tất cả các danh sách quản lý của hệ thống,
 - Hủy bỏ PCB của tiến trình.
- Hầu hết các hệ điều hành không cho phép các tiến trình con tiếp tục tồn tại nếu tiến trình cha đã kết thúc. Trong những hệ thống như thế, hệ điều hành sẽ tự động phát sinh một loạt các thao tác kết thúc tiến trình con.

Cấp phát tài nguyên cho tiến trình

- Về cơ bản, mỗi tài nguyên gồm những thông tin cơ bản sau:
 - Định danh tài nguyên
 - Trạng thái tài nguyên
 - Hàng đợi trên một tài nguyên
 - Bộ cấp phát
- Các mục tiêu của kỹ thuật cấp phát:
 - Bảo đảm một số lượng hợp lệ các tiến trình truy xuất đồng thời đến các tài nguyên không chia sẻ được
 - Cấp phát tài nguyên cho tiến trình có yêu cầu trong một khoảng thời gian trì hoãn có thể chấp nhận được
 - Tối ưu hóa việc sử dụng tài nguyên
- ❖ Để có thể thỏa mãn các mục tiêu kể trên, cần phải giải quyết các vấn đề nảy sinh khi có nhiều tiến trình đồng thời yêu cầu một tài nguyên không thể chia sẻ.

Tiểu trình

Định nghĩa

Nguyên lý hoạt động của tiểu trình

So sánh tiến trình và tiểu trình

Định nghĩa

- Trong một tiến trình có thể có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ, các dòng xử lý này hoạt động song song tương tự như các tiến trình phân biệt (ngoại trừ việc chia sẻ không gian địa chỉ).
- Mỗi dòng xử lý này gọi là một tiểu trình; đây là một đơn vị xử lý cơ bản trong hệ thống.

Nguyên lý hoạt động của tiểu trình

- Một tiểu trình xử lý tuần tự đoạn code của nó, sở hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng.
- Các tiểu trình chia sẻ CPU với nhau giống như cách chia sẻ giữa các tiến trình: Một tiểu trình xử lý trong khi các tiểu trình khác chờ đến lượt.
- Một tiểu trình cũng có thể tạo lập các tiến trình con, và nhận các trạng thái khác nhau như một tiến trình thực sự.
- Một tiến trình có thể sở hữu nhiều tiểu trình.
- Các tiểu trình trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung, điều này có nghĩa là các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình. Một tiểu trình có thể truy xuất đến cả các stack của những tiểu trình khác trong cùng tiến trình.

So sánh tiến trình và tiểu trình

Phân bổ thông tin lưu trữ:

- Tiến trình: Không gian địa chỉ, tài nguyên toàn cục, các thông tin thống kê.
- Tiểu trình: Con trỏ lệnh các thanh ghi stack, tài nguyên cục bộ.

Điều phối tiến trình

Giới thiệu

Các chiến lược điều phối

Chiến lược điều phối FIFO

Chiến lược điều phối xoay vòng

Chiến lược điều phối với độ ưu tiên

Chiến lược điều phối theo công việc
ngắn nhất

Giới thiệu

- Trong môi trường đa chương, có thể xảy ra tình huống nhiều tiến trình đồng thời sẵn sàng để xử lý. Mục tiêu của các hệ phân chia thời gian (time sharing) là chuyển đổi CPU qua lại giữa các tiến trình một cách thường xuyên để nhiều người sử dụng có thể tương tác cùng lúc với từng chương trình trong quá trình xử lý.
- Để thực hiện được mục tiêu này, hệ điều hành phải lựa chọn tiến trình được xử lý tiếp theo. Bộ điều phối sẽ sử dụng một giải thuật điều phối thích hợp để thực hiện nhiệm vụ này. Một thành phần khác của hệ điều hành cũng tiềm ẩn trong công tác điều phối là bộ phân phối (dispatcher). Bộ phân phối sẽ chịu trách nhiệm chuyển đổi ngữ cảnh và trao CPU cho tiến trình được chọn bởi bộ điều phối để xử lý.

- Mục tiêu điều phối: sự công bằng, tính hiệu quả, thời gian đáp ứng hợp lý, thời gian lưu lại trong hệ thống, thông lượng tối đa.
- Các đặc điểm của tiến trình: tính hướng xuất/nhập, tính hướng xử lý, tiến trình tương tác hay xử lý theo lô, độ ưu tiên của tiến trình, thời gian đã sử dụng CPU của tiến trình, thời gian còn lại tiến trình cần để hoàn tất.
- Điều phối độc quyền và điều phối không độc quyền.

Điều phối độc quyền

Nguyên lý điều phối độc quyền cho phép một tiến trình khi nhận được CPU sẽ có quyền độc chiếm CPU đến khi hoàn tất xử lý hoặc tự nguyện giải phóng CPU. Khi đó quyết định điều phối CPU sẽ xảy ra trong các tình huống sau:

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa (blocked).
- Khi tiến trình kết thúc

Các giải thuật điều phối độc quyền thường dễ cài đặt. Tuy nhiên chúng thường không thích hợp với các hệ thống tổng quát nhiều người dùng, vì nếu cho phép một tiến trình có quyền xử lý bao lâu tùy ý, có nghĩa là tiến trình này có thể giữ CPU một thời gian không xác định, có thể ngăn cản những tiến trình còn lại trong hệ thống có một cơ hội để xử lý.

Điều phối không độc quyền

Điều phối theo nguyên lý không độc quyền cho phép tạm dừng hoạt động của một tiến trình đang sẵn sàng xử lý. Khi một tiến trình nhận được CPU, nó vẫn được sử dụng CPU đến khi hoàn tất hoặc tự nguyện giải phóng CPU, nhưng một tiến trình khác có độ ưu tiên có thể dành quyền sử dụng CPU của tiến trình ban đầu. Như vậy là tiến trình có thể bị tạm dừng hoạt động bất cứ lúc nào mà không được báo trước, để tiến trình khác xử lý. Các quyết định điều phối xảy ra khi:

- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái bị khóa (ví dụ chờ một thao tác nhập/xuất hay chờ một tiến trình con kết thúc)
- Khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready (ví dụ xảy ra một ngắn)
- Khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready (ví dụ thao tác nhập/xuất hoàn tất).
- Khi tiến trình kết thúc.

Đồng hồ ngắt giờ

- Khi một tiến trình được cấp CPU, nó sẽ thực hiện các xử lý, tuy nhiên hoạt động của nó có thể ảnh hưởng đến toàn bộ hệ thống. Để tránh các tiến trình của người sử dụng độc chiếm CPU và gây tác hại đến hệ thống (dù vô tình hay cố ý), hệ điều hành phải xây dựng những cơ chế thu hồi lại CPU từ các tiến trình của người dùng một cách chủ động.
- Hệ điều hành sử dụng một bộ đếm thời gian/đồng hồ ngắt giờ và quy định một thông số thời gian t thích hợp ứng với một lượt cấp phát CPU cho một tiến trình. Khi hệ thống vận hành, cứ sau khoảng thời gian t sẽ xảy ra một ngắt báo hiệu hết thời gian xử lý của tiến trình hiện hành. Khi đó hệ điều hành sẽ được kích hoạt, và bộ điều phối sẽ quyết định tiến trình nào sẽ được cấp phát CPU trong lượt kế tiếp.
- Cơ chế ngắt giờ cho phép thực hiện được sự tương tác tự nhiên với nhiều người dùng, đồng thời ngăn chặn được tình trạng hệ thống bị quẩn trong một tiến trình của người sử dụng với vòng lặp vô hạn.

Độ ưu tiên của tiến trình

- Độ ưu tiên tĩnh:

Là độ ưu tiên được gán sẵn cho tiến trình, và không thay đổi bất kể sự biến động của môi trường. Cơ chế ưu tiên tĩnh dễ thực hiện nhưng đôi khi không hợp lý vì môi trường thay đổi có thể ảnh hưởng đến tầm quan trọng của tiến trình.

- Độ ưu tiên động:

Là độ ưu tiên thay đổi theo thời gian và môi trường xử lý của tiến trình. Tiến trình được khởi động với một độ ưu tiên, độ ưu tiên này thường chỉ giữ giá trị trong một khoảng thời gian ngắn, sau đó hệ thống sẽ sửa đổi giá trị độ ưu tiên trong từng giai đoạn thực hiện của tiến trình cho phù hợp với tình hình hệ thống.

Các chiến lược điều phối tiến trình

- Chiến lược điều phối FIFO
- Chiến lược điều phối xoay vòng
- Chiến lược điều phối với độ ưu tiên
- Chiến lược điều phối theo công việc **ngắn nhất**

Chiến lược điều phối FIFO

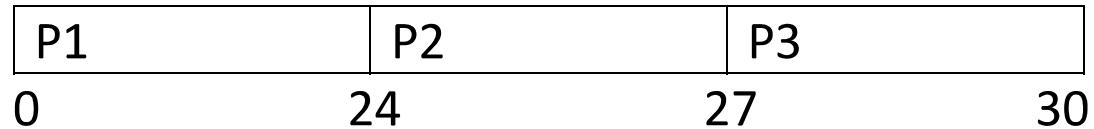
- CPU được cấp phát cho tiến trình đầu tiên trong danh sách sẵn sàng có yêu cầu, là tiến trình được đưa vào hệ thống sớm nhất.
- Đây là thuật toán điều phối theo nguyên tắc độc quyền.
- Một khi CPU được cấp phát cho tiến trình, CPU chỉ được tiến trình tự nguyện giải phóng khi kết thúc xử lý hay khi có một yêu cầu nhập/xuất.

Ví dụ 1.

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

Thứ tự cấp phát CPU cho các tiến trình là: P1,P2,P3

Biểu đồ Gantt:



***Thời gian đáp ứng:** Là thời gian khi tiến trình đó vào hàng đợi cho đến khi lần đầu tiên nó chạy.

P1: $0 - 0 = 0$

P2: $24 - 1 = 23$

P3: $27 - 2 = 25$

Thời gian trung bình = $(0+23+25)/3=16\text{ms}$

***Thời gian hoàn thành:** Là thời gian kể từ lúc tiến trình đó vào hàng đợi đến khi xử lý xong. Thời gian kết thúc của tiến trình đó trừ thời gian đến.

P1: $24-0=24$

P2: $27-1=26$

P3: $30-2=28$

Thời gian trung bình: 26

***Thời gian chờ (chờ nhận CPU):** Thời gian hoàn thành-thời gian sử dụng CPU:

P1: $24-24=0$

P2: $26-3=23$

P3: $28-3=25$

Thời gian chờ trung bình: 16.

Thảo luận

- Thời gian chờ trung bình không đạt cực tiểu và biến đổi đáng kể đối với các giá trị về thời gian yêu cầu xử lý và thứ tự khác nhau của các tiến trình trong danh sách sẵn sàng.
- Có thể xảy ra hiện tượng tích lũy thời gian chờ, khi tất cả các tiến trình (có thể có yêu cầu thời gian ngắn) phải chờ đợi một tiến trình có yêu cầu thời gian dài kết thúc xử lý.
- Giải thuật này không phù hợp với các hệ phân chia thời gian, trong các hệ này, cần cho phép mỗi tiến trình được cấp phát CPU đều đặn trong từng khoảng thời gian.
- Chiến lược này hay còn được gọi là chiến lược FCFS (FIRST COME FIRST SERVED)

Chiến lược điều phối xoay vòng (Round robin)

- Danh sách sẵn sàng được xử lý như một danh sách vòng, bộ điều phối lần lượt cấp phát cho từng tiến trình trong danh sách một khoảng thời gian sử dụng CPU gọi là *quantum*.
- Đây là một giải thuật điều phối không độc quyền:
 - Khi một tiến trình sử dụng CPU đến hết thời gian *quantum* dành cho nó, hệ điều hành thu hồi CPU và cấp cho tiến trình kế tiếp trong danh sách.
 - Nếu tiến trình bị khóa hay kết thúc trước khi sử dụng hết thời gian *quantum*, hệ điều hành cũng lập tức cấp phát CPU cho tiến trình khác.
 - Khi tiến trình tiêu thụ hết thời gian CPU dành cho nó mà chưa hoàn tất, tiến trình được đưa trở lại vào cuối danh sách sẵn sàng để chờ đợi được cấp CPU trong lượt kế tiếp.

Ví dụ 2.

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

- Nếu sử dụng $quantum=4\text{ms}$, thứ tự cấp phát CPU sẽ là:
(biểu đồ Gantt)

P1	P2	P3	P1	P1	P1	P1	P1
0	4	7	10	14	18	22	26 30

- Thời gian chờ trung bình sẽ là: $(0 + 6 + 3 + 5)/3 = 4.67\text{ms}$.
- Nếu có n tiến trình trong danh sách sẵn sàng và sử dụng $quantum q$, thì mỗi tiến trình sẽ được cấp phát CPU $1/n$ trong từng quãng thời gian q . Mỗi tiến trình sẽ không đợi quá $(n-1)/q$ đơn vị thời gian trước khi nhận được CPU cho lượt kế tiếp.

*Thời gian đáp ứng trung bình:

*Thời gian hoàn thành trung bình:

*Thời gian chờ trung bình:

Thảo luận

- Vấn đề đáng quan tâm đối với giải thuật Round robin là độ dài của *quantum* (thường là 10.. 100ms)
- Nếu thời lượng *quantum* quá bé sẽ phát sinh quá nhiều sự chuyển đổi giữa các tiến trình và khiến cho việc sử dụng CPU kém hiệu quả.
- Nhưng nếu sử dụng *quantum* quá lớn sẽ làm tăng thời gian hồi đáp và giảm khả năng tương tác của hệ thống.

Chiến lược điều phối với độ ưu tiên (Priority)

- Mỗi tiến trình được gán cho một độ ưu tiên tương ứng, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU đầu tiên.
- Độ ưu tiên có thể được định nghĩa nội tại hay nhờ vào các yếu tố bên ngoài.
- Độ ưu tiên nội tại sử dụng các đại lượng có thể đo lường để tính toán độ ưu tiên của tiến trình, ví dụ các giới hạn thời gian, nhu cầu bộ nhớ,...
- Độ ưu tiên cũng có thể được gán từ bên ngoài dựa vào các tiêu chuẩn do hệ điều hành như tầm quan trọng của tiến trình, loại người sử dụng sở hữu tiến trình,...

- Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hoặc không độc quyền. Khi một tiến trình được đưa vào danh sách các tiến trình sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện hành đang xử lý.
- Một giải thuật độc quyền sẽ chỉ đơn giản chèn tiến trình mới vào danh sách sẵn sàng, và tiến trình hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.
- Giải thuật điều phối với độ ưu tiên và không độc quyền sẽ thu hồi CPU từ tiến trình hiện hành để cấp phát cho tiến trình mới nếu độ ưu tiên của tiến trình này cao hơn tiến trình hiện hành.

Ví dụ 3. Chiến lược điều phối với độ ưu tiên (Priority)

P	Thời gian đến (arrival time)	Thời gian sử dụng CPU (service time)	Độ ưu tiên (Priority)
P1	0	24	3
P2	1	3	1
P3	2	3	2

Hãy tính thời gian đáp ứng trung bình, thời gian hoàn thành trung bình, thời gian chờ trung bình.

Giải:

Theo nguyên tắc độc quyền

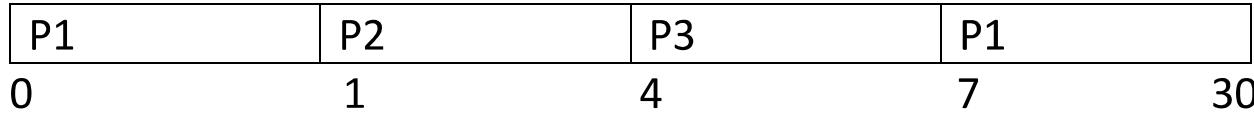
Giả sử cho độ ưu tiên của các tiến trình P1,P2,P3 lần lượt là 3,1,2; khi đó thứ tự cấp phát CPU như sau:

P1	P2	P3
0	24	27 30

(Tương tự như FIFO)

Theo nguyên không độc quyền

Biểu đồ Gantt (theo chế độ không độc quyền)



Thời gian đáp ứng:

$$P1: 0 - 0 = 0$$

$$P2: 1 - 1 = 0$$

$$P3: 4 - 2 = 2$$

Thời gian hoàn thành:

$$P1: 30 - 0 = 30$$

$$P2: 4 - 1 = 3$$

$$P3: 7 - 2 = 5$$

Thời gian chờ (thời gian hoàn thành – thời gian sử dụng CPU)

$$P1: 30 - 24 = 6$$

$$P2: 3 - 3 = 0$$

$$P3: 5 - 3 = 2$$

*Lấy trung bình cộng,... suy ra kết quả.

Thảo luận

- Tình trạng “đói CPU” (starvation) là một vấn đề chính yếu của giải thuật sử dụng độ ưu tiên.
- Các giải thuật này có thể để các tiến trình có độ ưu tiên thấp chờ đợi CPU vô hạn! Để giải quyết vấn đề các tiến trình có độ ưu tiên cao chiếm dụng CPU vô thời hạn, bộ điều phối sẽ giảm dần độ ưu tiên của các tiến trình này sau mỗi ngắt đồng hồ.
- Nếu độ ưu tiên của tiến trình này giảm xuống thấp hơn tiến trình có độ ưu tiên cao thứ nhì, sẽ xảy ra sự chuyển đổi quyền sử dụng CPU. Quá trình này gọi là sự “lão hóa” tiến trình (aging).

Chiến lược điều phối theo công việc ngắn nhất (Shortest Job first – SJF)

- Đây là một trường hợp đặc biệt của giải thuật điều phối với độ ưu tiên. Trong giải thuật này, độ ưu tiên p được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý t mà tiến trình yêu cầu: $p=1/t$. Khi CPU được tự do, nó sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc; tiến trình ngắn nhất.
- Giải thuật này cũng có thể độc quyền hay không độc quyền. Sự chọn lựa xảy ra khi có một tiến trình mới được đưa vào danh sách sẵn sàng trong khi một tiến trình khác đang xử lý. Tiến trình mới có thể sở hữu một yêu cầu thời gian sử dụng CPU cho lần tiếp theo (CPU burst) ngắn hơn thời gian còn lại mà tiến trình hiện hành cần xử lý. Giải thuật SJF không độc quyền sẽ dừng hoạt động của tiến trình hiện hành, trong khi giải thuật độc quyền sẽ cho phép tiến trình hiện hành tiếp tục xử lý.

Ví dụ 4a.

Tiến trình	Thời điểm vào	Thời gian xử lý
P1	0	6
P2	0	8
P3	0	7
P4	0	3

Q1. Sử dụng giải thuật SJF độc quyền, thứ tự cấp phát CPU như sau:



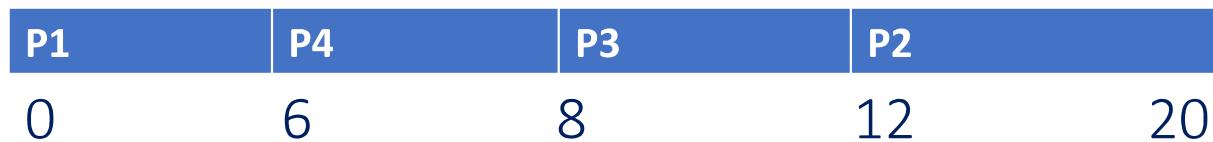
Q2. Tìm thời gian chờ trung bình của các tiến trình ? (Đáp số:7)

Q3. Nếu sử dụng điều phối FCFS thì thời gian chờ trung bình là bao nhiêu ?
(Đáp số: 10.25)

Ví dụ 4b.

P	Thời gian đến	Thời gian sử dụng CPU
P1	0	6
P2	1	8
P3	2	4
P4	3	2

Sử dụng giải thuật SJF độc quyền, thứ tự cấp phát CPU như sau:



Ví dụ 4b. (tiếp theo)

- Sử dụng giải thuật SJF không độc quyền (Shortest Remaining Time first - SRTF), thứ tự cấp phát CPU như sau:



(Lưu ý tại thời điểm 2, thời gian sử dụng CPU còn lại của P1 và P3 đều bằng 4, nên vẫn cho P1 làm tiếp đến thời điểm 3).

Ví dụ 4c.

Sử dụng giải thuật SJF không độc quyền

P	Thời gian đến	Thời gian sử dụng CPU
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Q1. Tìm thứ tự cấp phát CPU ?

Q2. Tìm thời gian chờ trung bình: (Đáp số: 6.5 ms)

Q3. Nếu với bảng dữ liệu trên mà dùng SJF độc quyền thì thời gian chờ trung bình là bao nhiêu ? (Đáp số 7.75 ms)

Thảo luận

Giải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu. Khó khăn thực sự của giải thuật SJF là không thể biết được thời gian yêu cầu xử lý còn lại của tiến trình mà chỉ có thể dự đoán giá trị này theo cách tiếp cận sau: Gọi t_n là độ dài của độ dài xử lý lần thứ n , τ_{n+1} là giá trị dự đoán cho lần xử lý tiếp theo. Với hy vọng giá trị dự đoán sẽ gần giống với các giá trị trước đó, có thể sử dụng công thức:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

Trong công thức này, t_n chứa đựng thông tin gần nhất; τ_n chứa đựng các thông tin quá khứ được tích lũy. Tham số α ($0 \leq \alpha \leq 1$) kiểm soát trọng số của hiện tại gần hay quá khứ ảnh hưởng đến công thức dự đoán.

Thảo luận...

Thông thường một tiến trình sẽ được gán vĩnh viễn với một danh sách ở cấp độ ưu tiên i khi nó được đưa vào hệ thống. Các tiến trình không di chuyển giữa các danh sách. Cách tổ chức này sẽ làm giảm chi phí điều phối, nhưng lại thiếu linh động và có thể dẫn đến tình trạng ‘đói CPU’ cho các tiến trình thuộc về những danh sách có độ ưu tiên thấp. Do vậy có thể xây dựng giải thuật điều phối nhiều cấp ưu tiên và xoay vòng.

Giải thuật này sẽ chuyển dần một tiến trình từ danh sách có độ ưu tiên cao xuống danh sách có độ ưu tiên thấp hơn sau mỗi lần sử dụng CPU. Cũng vậy, một tiến trình chờ quá lâu trong các danh sách có độ ưu tiên thấp cùng có thể được chuyển dần lên các danh sách có độ ưu tiên cao hơn.

Khi xây dựng một giải thuật điều phối nhiều cấp ưu tiên và xoay vòng cần quyết định các tham số:

- Số lượng các cấp ưu tiên
- Giải thuật điều phối cho từng danh sách ứng với mỗi cấp ưu tiên.
- Phương pháp xác định thời điểm di chuyển một tiến trình lên danh sách có độ ưu tiên cao hơn.
- Phương pháp xác định thời điểm di chuyển một tiến trình xuống danh sách có độ ưu tiên thấp hơn.
- Phương pháp sử dụng để xác định một tiến trình mới được đưa vào hệ thống sẽ thuộc danh sách ứng với độ ưu tiên nào.

Đồng bộ hóa tiến trình

1. Dẫn nhập

2. Các cơ chế thông tin liên lạc

3. Đồng bộ hóa tiến trình

1. Dẫn nhập

Nhu cầu liên lạc giữa các tiến trình:

- Trong môi trường đa chương, một tiến trình không đơn độc trong hệ thống, mà có thể ảnh hưởng đến các tiến trình khác, hoặc bị các tiến trình khác tác động.
- Nói cách khác, các tiến trình là những thực thể độc lập, nhưng chúng vẫn có nhu cầu liên lạc với nhau để:
 - Chia sẻ thông tin
 - Hợp tác hoàn thành tác vụ

Các vấn đề nảy sinh trong việc liên lạc giữa các tiến trình

Do mỗi tiến trình sở hữu một không gian địa chỉ riêng biệt, nên các tiến trình không thể liên lạc trực tiếp dễ dàng mà phải nhờ vào các cơ chế do hệ điều hành cung cấp. Khi cung cấp cơ chế liên lạc cho các tiến trình, hệ điều hành phải tìm giải pháp cho các vấn đề chính yếu sau:

- Liên kết tường minh hay liên kết tiềm ẩn
- Liên lạc theo chế độ đồng bộ hay không đồng bộ
- Liên lạc giữa các tiến trình trong hệ thống tập trung hay hệ thống phân tán ?

Hầu hết các hệ điều hành đưa ra nhiều cơ chế liên lạc khác nhau, mỗi cơ chế có những đặc tính riêng, và thích hợp trong một hoàn cảnh chuyên biệt.

2.Các cơ chế thông tin liên lạc

❑ Tín hiệu (signal):

- Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình. Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra. Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng.
- Mỗi tiến trình sở hữu một bảng biểu diễn các tín hiệu khác nhau. Với mỗi tín hiệu sẽ có tương ứng một trình xử lý tín hiệu quy định các xử lý của tiến trình khi nhận được tín hiệu tương ứng.
- Liên lạc bằng tín hiệu mang tính chất không đồng bộ, nghĩa là một tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tín hiệu. Hơn nữa các tiến trình không thể kiểm tra được sự kiện tương ứng với tín hiệu có thật sự xảy ra ? Cuối cùng, các tiến trình chỉ có thể thông báo cho nhau về một biến cố nào đó, mà không trao đổi dữ liệu theo cơ chế này được.

❑ Pipe:

- Một pipe là một kênh liên lạc trực tiếp giữa hai tiến trình: dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte.
- Khi một pipe được thiết lập giữa hai tiến trình, một trong chúng sẽ ghi dữ liệu vào pipe và tiến trình kia sẽ đọc dữ liệu từ pipe. Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO. Một pipe có kích thước giới hạn (thường là 4096 ký tự).

- Một tiến trình chỉ có thể sử dụng một pipe do nó tạo ra hay kế thừa từ tiến trình cha. Hệ điều hành cung cấp các lời gọi hệ thống read/write cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe. Hệ điều hành cũng chịu trách nhiệm đồng bộ hóa việc truy xuất pipe trong các tình huống:
 - Tiến trình đọc pipe sẽ bị khóa nếu pipe trống, nó sẽ phải đợi đến khi pipe có dữ liệu để truy xuất.
 - Tiến trình ghi pipe sẽ bị khóa nếu pipe đầy, nó sẽ phải đợi đến khi pipe có chỗ trống để chứa dữ liệu.
- Liên lạc bằng pipe là một cơ chế liên lạc một chiều; nghĩa là một tiến trình chỉ có thể thực hiện được một trong hai thao tác là đọc hoặc ghi. Một số hệ điều hành cho phép thiết lập hai pipe giữa một cặp tiến trình để tạo liên lạc hai chiều; trong những hệ thống đó có nguy cơ xảy ra tình trạng tắc nghẽn (deadlock). Một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha con, và trên cùng một máy tính.

❑ Vùng nhớ chia sẻ

- Cách tiếp cận của cơ chế này là cho nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là vùng nhớ chia sẻ (share memory). Không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây, dữ liệu chỉ đơn giản được đặt vào một vùng nhớ mà nhiều tiến trình có thể truy cập được.
- Với phương thức này, các tiến trình chia sẻ một vùng nhớ vật lý thông qua không gian địa chỉ của chúng. Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình, và khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết nối vùng nhớ chung đó vào không gian địa chỉ của từng tiến trình, và thao tác trên đó như một vùng nhớ riêng của mình.

Trao đổi thông điệp

- Hệ điều hành cung cấp một cơ chế liên lạc giữa các tiến trình không thông qua việc chia sẻ một tài nguyên chung, mà thông qua việc gửi thông điệp. Để hỗ trợ cơ chế liên lạc bằng thông điệp, hệ điều hành cung cấp các hàm IPC chuẩn (Interprocess communication), cơ bản là 2 hàm: send (message) và Receive(message)
- Đơn vị truyền thông tin trong cơ chế trao đổi thông điệp là một thông điệp, do đó các tiến trình có thể trao đổi dữ liệu ở dạng có cấu trúc.

❑ Sockets

- Một socket là một thiết bị truyền thông hai chiều tương tự như tập tin, chúng ta có thể đọc hay ghi lên nó, tuy nhiên mỗi socket là một thành phần trong một mối nối nào đó giữa các máy trên mạng máy tính và các thao tác đọc/ghi chính là sự trao đổi dữ liệu giữa các ứng dụng trên nhiều máy khác nhau.
- Sử dụng socket có thể mô phỏng 2 phương thức liên lạc trong thực tế: liên lạc thư tín (socket đóng vai trò bưu cục) và liên lạc điện thoại (socket đóng vai trò tổng đài).
- Cơ chế socket có thể sử dụng để chuẩn hóa mối liên lạc giữa các tiến trình vốn không liên hệ với nhau, và có thể hoạt động trong những hệ thống khác nhau.

3.Đồng bộ hóa tiến trình

- Tổng quan
- Giải pháp đồng bộ hóa
- Các vấn đề cổ điển của đồng bộ hóa

Tổng quan

- Nhu cầu đồng bộ hóa: trong một hệ thống cho phép các tiến trình liên lạc với nhau, bao giờ hệ điều hành cũng cần cung cấp kèm theo những cơ chế đồng bộ hóa vì các lý do sau:
 - Yêu cầu độc quyền truy xuất
 - Yêu cầu phối hợp

Yêu cầu độc quyền truy xuất

Các tài nguyên trong hệ thống được phân thành hai loại: tài nguyên có thể chia sẻ cho phép nhiều tiến trình đồng thời truy xuất, và tài nguyên không thể chia sẻ chỉ chấp nhận một (hay một số lượng hạn chế) tiến trình sử dụng tại một thời điểm. Tính không thể chia sẻ của tài nguyên thường có nguồn gốc từ một trong hai nguyên nhân sau đây:

- Đặc tính cấu tạo phần cứng của tài nguyên không cho phép chia sẻ
- Nếu nhiều tiến trình sử dụng tài nguyên đồng thời, có nguy cơ xảy ra các kết quả không dự đoán được do hoạt động của các tiến trình trên tài nguyên ảnh hưởng lẫn nhau.
- Để giải quyết vấn đề, cần đảm bảo tiến trình độc quyền truy xuất tài nguyên, nghĩa là hệ thống phải kiểm soát sao cho tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.

Yêu cầu phối hợp

Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như lần xuất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát bộ xử lý... có thể nói rằng các tiến trình hoạt động không đồng bộ với nhau. Nhưng có những tình huống các tiến trình cần hợp tác trong việc hoàn thành tác vụ, khi đó cần phải đồng bộ hóa hoạt động của các tiến trình, ví dụ một tiến trình chỉ có thể xử lý nếu một tiến trình khác đã kết thúc công việc nào đó,...

Miền găng

- Vấn đề tranh đoạt điều khiển (race condition)

Khi có nhiều hơn hai tiến trình đọc và ghi dữ liệu trên cùng một vùng nhớ chung, và kết quả phụ thuộc vào sự điều phối tiến trình của hệ thống được gọi là các tình huống tranh đoạt điều khiển.

- Miền găng: Để ngăn chặn các tình huống lỗi có thể nảy sinh khi các tiến trình truy xuất đồng thời một tài nguyên không thể chia sẻ, cần phải áp đặt một sự truy xuất độc quyền trên tài nguyên đó: Khi một tiến trình đang sử dụng tài nguyên, thì những tiến trình khác không được truy xuất đến tài nguyên đó. Đoạn chương trình trong đó có khả năng xảy ra các mâu thuẫn truy xuất trên tài nguyên chung được gọi là miền găng (critical section)

Giải quyết bài toán miền găng

Có thể giải quyết vấn đề mâu thuẫn truy xuất nếu có thể bảo đảm tại một thời điểm chỉ có duy nhất một tiến trình được xử lý lệnh trong miền găng. Một phương pháp giải quyết tốt bài toán miền găng cần thỏa mãn 4 điều kiện sau:

- o Không có hai tiến trình cùng lúc ở trong miền găng.
- o Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý trong hệ thống
- o Một tiến trình tạm dừng bên ngoài miền găng không được ngăn cản các tiến trình khác vào miền găng
- o Không có tiến trình nào phải chờ vô hạn để được vào miền găng.

Giải pháp đồng bộ hóa

Có nhiều giải pháp để thực hiện việc truy xuất độc quyền, các giải pháp này được phân biệt thành hai lớp tùy theo cách tiếp cận trong xử lý của tiến trình bị khóa:

- Giải pháp “busy waiting”
- Giải pháp “SLEEP and WAKEUP”

(sinh viên nghiên cứu thêm trong tài liệu tham khảo)

Các vấn đề cổ điển của đồng bộ hóa

- Vấn đề người sản xuất-người tiêu thụ (Producer-Consumer)
- Mô hình Readers-Writers
(sinh viên nghiên cứu thêm trong tài liệu tham khảo)

Tắc nghẽn (deadlock)

1. Định nghĩa

2. Điều kiện xuất hiện các tắc nghẽn

3. Tránh tắc nghẽn

4. Phát hiện và hiệu chỉnh tắc nghẽn

1. Định nghĩa tắc nghẽn

- Một tập hợp các tiến trình được định nghĩa ở trong tình trạng tắc nghẽn (deadlock) khi mỗi tiến trình trong tập hợp đều chờ đợi một sự kiện mà chỉ có một tiến trình khác trong tập hợp có thể phát sinh được.
- Nói cách khác, mỗi tiến trình trong tập hợp đều chờ được cấp phát một tài nguyên hiện đang bị một tiến trình khác cũng ở trạng thái block 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.
- (cũng có thể phát biểu ngắn như sau: “Một tiến trình gọi là deadlock nếu nó đang đợi một sự kiện mà sẽ không bao giờ xảy ra”).

2.Điều kiện xuất hiện tắc nghẽn

Điều kiện xuất hiện các tắc nghẽn

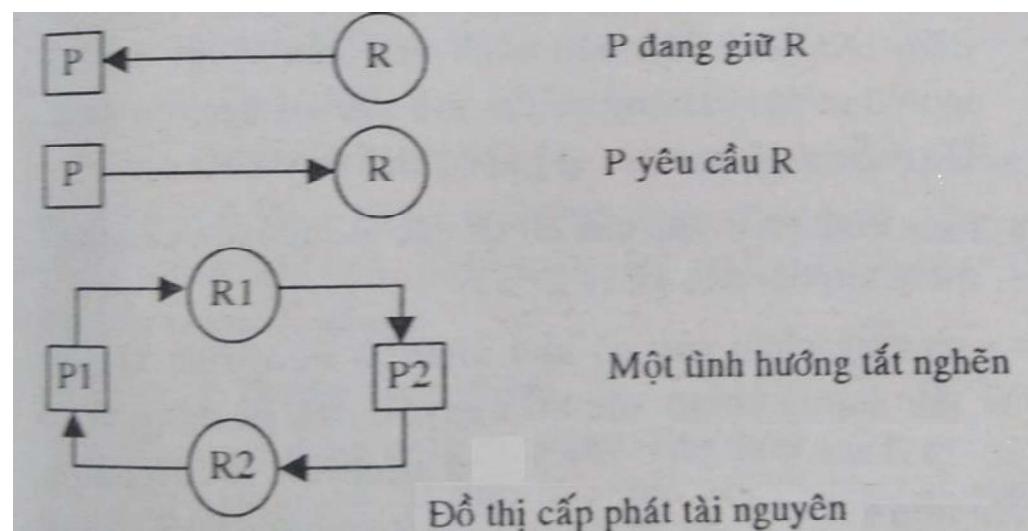
Sau đây là các điều kiện cần có thể làm xuất hiện tắc nghẽn:

- 1) Khi sử dụng tài nguyên không thể chia sẻ
- 2) Sự chiếm giữ và yêu cầu thêm tài nguyên
- 3) Không thu hồi tài nguyên từ tiến trình đang giữ chúng
- 4) Tồn tại một chu kỳ trong đồ thị cấp phát tài nguyên.

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.

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 vuông, và mỗi tài nguyên được biểu diễn bằng hình tròn.



Các phương pháp xử lý tắc nghẽn

Chủ yếu có 3 hướng tiếp cận để xử lý tắc nghẽn:

- Sử dụng một nghi thức (protocol) để bảm đảm rằng hệ thống không bao giờ xảy ra tắc nghẽn.
- Cho phép xảy ra tắc nghẽn và tìm cách sửa chữa tắc nghẽn.
- Hoàn toàn bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn.

Ngăn chặn tắc nghẽn

Để tắc nghẽn không xảy ra, cần bảo đảm tối thiểu một trong 4 điều kiện không xảy ra

- 1) Tài nguyên không thể chia sẻ
 - 2) Sự chiếm giữ và yêu cầu thêm tài nguyên
 - 3) Không thu hồi tài nguyên
 - 4) Tồn tại một chu kỳ
- (SV nghiên cứu thêm trong tài liệu tham khảo)

3.Tránh tắc nghẽn

Ngăn cản tắc nghẽn là 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

- Chuỗi an toàn

Một chuỗi quá trình $\langle P_1, P_2, \dots, P_n \rangle$ là một *chuỗi an toàn* nếu với mọi $i = 1, \dots, n$, yêu cầu tối đa về tài nguyên của P_i có thể được thỏa bởi

- ✓ tài nguyên mà hệ thống đang có sẵn sàng (available)
- ✓ cùng với tài nguyên mà tất cả P_j , $j < i$, đang giữ.

- Trạng thái an toàn

Một trạng thái của hệ thống được gọi là *an toàn* (safe) nếu tồn tại một *chuỗi an toàn* (safe sequence).

- Trạng thái không an toàn

Một trạng thái của hệ thống được gọi là *không an toàn* (unsafe) nếu không tồn tại một chuỗi an toàn.

Giải thuật xác định trạng thái an toàn (banker)

Thuật toán banker dạng 1 (tìm bảng Need)

• Thuật toán trạng thái an toàn:

- 1) Khởi tạo Work=Available và Finish[i]=false $\forall i=1..n$
- 2) Tìm i sao cho Finish [i]==false và Need[i] \leq Work
 - Nếu không tìm được i, chuyển đến bước 4
- 3) Work = Work + Allocation [i], Finish [i]=true
 - Chuyển đến bước 2
- 4) Nếu Finish [i] == true $\forall i$ thì hệ thống ở trạng thái an toàn
- Độ phức tạp tính toán của thuật toán trạng thái an toàn: $O(m.n^2)$

Thuật toán banker

- Ví dụ: hệ thống gồm
 - 5 tiến trình P0, P1, P2, P3, P4
 - 3 loại tài nguyên A, B, C.
 - A có 10 thể hiện
 - B có 5 thể hiện
 - C có 7 thể hiện.
- Giả sử trạng thái của hệ thống tại thời điểm T0:

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

Tại thời điểm T0, hệ thống có
ở trạng thái an toàn hay không

Need		
A	B	C

Tiến trình	Work		
	A	B	C
	3	3	2

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

If $\text{Need} \leq \text{Work}$ thì chúng ta cộng dồn $\text{Work} = \text{Work} + \text{Allocation}$

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

Tiến trình	Work		
	A	B	C
	3	3	2
P1	5	3	2
P3	7	4	3
P4	7	4	5
P0	7	5	5
P2	10	5	7

⇒ Kết luận: hệ thống trên ở trạng thái an toàn vì nó tồn tại thứ tự an toàn là P1, P3, P4, P0, P2

Thuật toán banker dạng 2 (có bảng Request)

- 1) Giả sử Work và Finish là các vector m và n thành phần.
- Khởi tạo Work = Available
- Với mỗi $i=0..n-1$
 - nếu Allocation [i] $\neq 0$ gán Finish [i]=false
 - ngược lại gán Finish [i]=true
- 2) Tìm i sao cho Finish[i]==false và Request[i] \leq Work. Nếu không tìm thấy i , chuyển đến bước 4
- 3) Work = Work + Allocation, Finish[i] = true; chuyển đến bước 2
- 4) Nếu Finish[i]==false với $0 \leq i \leq n-1$ thì hệ thống đang bị bế tắc (và tiến trình Pi đang bế tắc).
- Độ phức tạp tính toán của thuật toán: $O(m.n^2)$

Ví dụ: 5 tiến trình P0, P1, P2, P3, P4
3 loại tài nguyên A, B, C.

- A có 7 thể hiện
- B có 2 thể hiện
- C có 6 thể hiện.
- Trạng thái cấp phát tài nguyên tại thời điểm T0:

	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			

Tiến trình	Work		
	A	B	C
	0	0	0
P0	0	1	0
P2	3	1	3
P3	5	2	4
P4	5	2	6
P1	7	2	6

Request \leq Work \rightarrow Work = Work + Allocation

\Rightarrow Kết luận: Hệ thống không bị deadlock

(Hệ thống an toàn) vì tồn tại thứ tự an toàn là
P0, P2, P3, P4, P1

Thuật toán banker dạng 3 (có yêu cầu thêm tài nguyên)

Thuật toán banker

- Ví dụ: hệ thống gồm

- 5 tiến trình P0, P1, P2, P3, P4
- 3 loại tài nguyên A, B, C.
 - A có 10 thể hiện
 - B có 5 thể hiện
 - C có 7 thể hiện.

- Giả sử trạng thái của hệ thống tại thời điểm T0:

Giả sử P1 yêu cầu cấp phát (1,0,2). Yêu cầu này có thể được thỏa mãn hay không?
Tương tự với:
P4(3,3,0)
P0(0,2,0)
P3(0,2,1)

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

Giả sử tiến trình P1 yêu cầu thêm (1,0,2) thể hiện. Cần cập nhật lại trạng thái mới thì xử lý thế nào ?

-Cập nhật lại Allocation của P1 là (3,0,2).

Điều kiện:

Request[1](1,0,2) \leq Available (3,3,2) -> True

Request[1](1,0,2) \leq Need[1] (1,2,2) -> True

-Nếu một trong hai điều kiện này sai thì kết luận việc yêu cầu cấp phát thêm là không thành công.

-Nếu cả hai điều kiện đều đúng thì tiến hành tính 2 ma trận Need và Work

-Tạo bảng Need: Giữ nguyên P0,P2,3,P4; chỉ cập nhật Need của P1

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

Tiến trình	Work		
	A	B	C
P1	2	3	0
P3	5	3	2
P4	7	4	3
P0	7	4	5
P2	10	5	7

Lưu ý cập nhật dòng này

Tồn tại chuỗi an toàn P1,P3,P4,P0,P2; tức yêu cầu cấp phát thêm theo đề bài là thành công

4. Phát hiện và hiệu chỉnh tắc nghẽn

Cần sử dụng các cấu trúc dữ liệu sau :

```
int Available[NumResources];
// Available[r] = số lượng các thẻ hiện còn tự do của
tài nguyên r
int Allocation[NumProcs, NumResources];
// Allocation[p, r] = số lượng tài nguyên r thực sự
cấp phát cho p
int Request[NumProcs, NumResources];
// Request[p, r] = số lượng tài nguyên r tiến trình p
yêu cầu thêm
```

- Giải thuật phát hiện tắc nghẽn

1. int Work[NumResources] = Available;
int Finish[NumProcs];
for (i = 0; i < NumProcs; i++)
 Finish[i] = (Allocation[i] == 0);
2. Tìm i sao cho
 a) Finish[i] == false
 b) Request[i] <= Work

Nếu không có i như thế, đến bước 4.

3. Work = Work + Allocation[i];
Finish[i] = true;
Đến bước 2

4. Nếu Finish[i] == true với mọi i,
thì hệ thống không có tắc nghẽn

Nếu Finish[i] == false với một số giá trị i,
thì các tiến trình mà Finish[i] == false sẽ ở trong
tình trạng tắc nghẽn.

Hiệu chỉnh tắc nghẽn

Khi đã phát hiện được tắc nghẽn, có hai lựa chọn chính để hiệu chỉnh tắc nghẽn:

- 1) Đinh chỉ hoạt động của các tiến trình liên quan
- 2) Thu hồi tài nguyên

Định chỉ hoạt động của các tiến trình liên quan

Cách này tiếp cận dựa trên việc thu hồi lại các tài nguyên của những tiến trình bị kết thúc. Có thể sử dụng một trong hai phương pháp sau:

- Định chỉ tất cả các tiến trình trong tình trạng tắc nghẽn
- Định chỉ từng tiến trình liên quan cho đến khi không còn chu trình gây tắc nghẽn: để chọn được tiến trình thích hợp bị định chỉ, phải dựa vào các yếu tố như độ ưu tiên, thời gian đã xử lý, số lượng tài nguyên đang chiếm giữ, số lượng tài nguyên yêu cầu,...

Thu hồi tài nguyên

Có thể hiệu chỉnh tắc nghẽn bằng cách thu hồi một số tài nguyên từ các tiến trình và cấp phát các tài nguyên này cho những tiến trình khác cho đến khi loại bỏ được chu trình tắc nghẽn. Cần giải quyết 3 vấn đề sau:

- Chọn lựa một “nạn nhân”: tiến trình nào sẽ bị thu hồi tài nguyên ? Và thu hồi những tài nguyên nào ?
- Trở lại trạng thái trước tắc nghẽn: khi thu hồi tài nguyên của một tiến trình, cần phải phục hồi trạng thái của tiến trình trở lại trạng thái gần nhất trước đó mà không xảy ra tắc nghẽn.
- Tình trạng “đói tài nguyên”: làm sao bảo đảm rằng không có một tiến trình luôn luôn bị thu hồi tài nguyên ?

Tóm tắt chương 2

- Tiến trình là một chương trình đang hoạt động
- Để sử dụng hiệu quả CPU, sự đa chương cần được đưa vào hệ thống
- Sự đa chương được tổ chức bằng cách lưu trữ nhiều tiến trình trong bộ nhớ tại một thời điểm, và điều phối CPU qua lại giữa các tiến trình trong hệ thống.
- Trong suốt chu trình sống, tiến trình chuyển đổi qua lại giữa các trạng thái ready, running và blocked.
- Mô hình đa tiêu trình cho phép mỗi tiến trình có thể tiến hành nhiều dòng xử lý đồng thời trong cùng một không gian địa chỉ nhằm thực hiện tác vụ hiệu quả hơn trong một số trường hợp.

-
- Bộ điều phối của hệ điều hành chịu trách nhiệm áp dụng một giải thuật điều phối thích hợp để chọn tiến trình thích hợp được sử dụng CPU, và bộ phân phối sẽ chuyển chuyển giao CPU cho tiến trình này.
 - Các giải thuật điều phối thông dụng như: FIFO, RR,SJF,...
 - Một tiến trình trong hệ thống có nhu cầu trao đổi thông tin để phối hợp hoạt động, do mỗi tiến trình có một không gian địa chỉ độc lập nên việc liên lạc chỉ có thể được thực hiện thông qua các cơ chế do hệ điều hành cung cấp.

-
- Miền găng là đoạn lệnh trong chương trình có khả năng phát sinh mâu thuẫn truy xuất. Để không xảy ra mâu thuẫn truy xuất, cần bảo đảm tại mỗi thời điểm chỉ có một tiến trình được vào miền găng.
 - Lập trình viên có thể sử dụng các cơ chế đồng bộ do hệ điều hành hay trình biên dịch trợ giúp.
 - Tắc nghẽn là tình trạng xảy ra trong một tập các tiến trình nếu có hai hay nhiều hơn các tiến trình chờ đợi vô hạn một sự kiện chỉ có thể được phát sinh bởi một tiến trình cũng đang chờ khác trong tập các tiến trình này.

Bài tập

1. Cài đặt các thuật toán điều phối tiến trình
2. Cài đặt thuật toán xác định trạng thái an toàn
3. Thực hành một số lệnh của window11 (theo từng nhóm lệnh)
4. Thực hành một số lệnh của Linux/Ubuntu(theo từng nhóm lệnh)

5.

- a. Một hệ thống có 3 ổ băng từ và 3 tiến trình P1, P2, P3 với trạng thái cấp phát tài nguyên ở thời điểm T_i thể hiện bằng véc-tơ Allocation = (1, 0, 1) và Max = (1, 2, 2). Hãy chứng minh trạng thái {P1, P2, P3} an toàn.
- b. Một hệ thống có 3 ổ băng từ và 3 tiến trình P1, P2, P3 với trạng thái cấp phát tài nguyên tại thời điểm T_i thể hiện bằng các véc-tơ Allocation=(0, 2, 1) và Max=(2,2, 2). Hãy chứng minh trạng thái P1, P2, P3 an toàn.
- c. Một hệ thống có 5 tiến trình với tình trạng tài nguyên như sau. Hãy chứng minh trạng thái {P0, P2, P3, P4, P1} an toàn. Xác định có nên đáp ứng yêu cầu (0, 4, 3, 0) của P1 ?

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P ₀	0	0	1	2	0	0	1	2	1	5	2	0
P ₁	1	0	0	0	1	7	5	0				
P ₂	1	3	5	4	2	3	5	6				
P ₃	0	6	3	2	0	6	5	2				
P ₄	0	0	1	4	0	6	5	6				

6.Giả sử tình trạng hiện hành của hệ thống được mô tả như sau:

	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	4	1	2
P2	6	1	3	2	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

Tiến trình P2 yêu cầu 4 cho R1, 1 cho R3.

Hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không ?

(đã có lời giải ở các slide kế; yêu cầu cấp phát đáp ứng được)

- Nhận thấy Available[1]=4, Available[3]=2 đủ để thỏa mãn yêu cầu của P2, ta có:



	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0	0	1	1
P2	0	0	1	6	1	2			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0	6	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	7	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	4
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	4	2	0	0	0	2			

	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	0	0	0			