Chương 2. Quản lý tiến trình

2.1. Tiến trình

2.1.1. Mô hình tiến trình

Chương trình máy tính chứa các chỉ thị điều khiển máy tính thực hiện một tác vụ nào đó; chương trình là một thực thể thụ động.

Tiến trình (process) là một chương trình đang thực hiện; tiến trình là một thực thể hoạt động.

Tiến trình là một chương trình đang xử lý, sở hữu một con trỏ lệnh, tập các thanh ghi và các biến. 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 và thiết bị nhập/xuất.

2.1.2. Các trạng thái của tiến trình

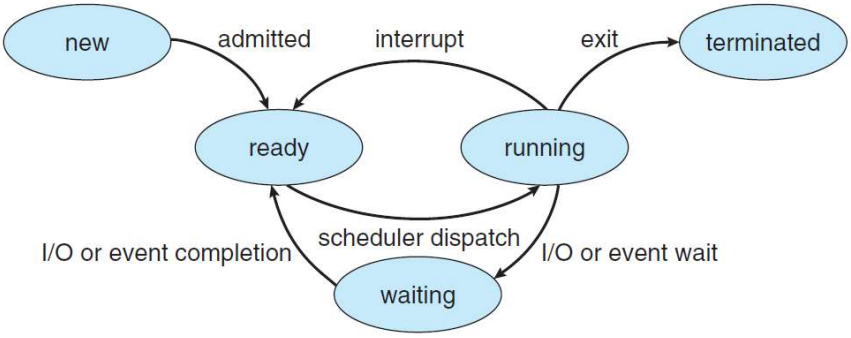
Trạng thái của tiến trình (process state) tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điểm đó. Trong quá trình sống của nó, một tiến trình thay đổi trạng thái do nhiều nguyên nhân như: phải chờ một sự kiện nào đó xảy ra, hay đợi một thao tác nhập/xuất hoàn tất, buộc phải dừng hoạt động do đã hết thời gian xử lý,…

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

Khởi tạo (*new*):	Tiến trình đang được khởi tạo
Thực hiện (*running*):	Các câu lệnh của tiến trình được xử lý
Chờ đợi (*waiting*/*blocked*):	Tiến trình chờ được cấp phát một tài nguyên, hay chờ một sự kiến xảy ra.
Sẵn sàng (*ready*):	Tiến trình chờ được cấp CPU để xử lý.
Kết thúc (*terminated*):	Tiến trình hoàn tất xử lý.

Sơ đồ chuyển trạng thái tiến trình

Hình 2.1. là sơ đồ chuyển trạng thái tiến trình (diagram of process state). Với hệ thống có một processor thì có duy nhất một tiến trình ở trạng thái *running*, có thể có nhiều tiến trình ở trạng thái *waiting*/ *blocked* hoặc *ready*.



Hình 2.1. Diagram of process state

Các cung chuyển tiếp trong sơ đồ trạng thái biểu diễn các sự chuyển trạng thái có thể xảy ra trong các điều kiện sau:

(*new, ready):*	Tiến trình mới tạo được đưa vào hệ thống.
(ready, running):	Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU.
(running, terminated):	Tiến trình kết thúc.
(running, waiting/blocked):	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.
(running, ready):	Bộ điều phối chọn một tiến trình khác để cho xử lý.
(waiting/blocked, ready):	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.

2.1.3. 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 và 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. 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.

2.1.4. 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 về: trạng thái tiến trình, bộ đếm lệnh, các thanh ghi của CPU, thông tin dùng để điều phối tiến trình, thông tin quản lý bộ nhớ, thông tin tài nguyên có thể sử dụng, thông tin thống kê, con trỏ trỏ đến một PCB khác,…

Hình 2.2. Process control block (PCB) [3]

Một số thành phần chủ yếu của PCB 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: 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.

2.1.5. Thao tác trên tiến trình

Hệ điều hành cung cấp các thao tác chủ yếu 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.

Trong mục này, giáo trình chỉ trình bày về hai thao tác tạo lập tiến trình và kết thúc 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 các 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 dạng 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 PCB cho tiến trình
- Cấp phát các tài nguyên ban đầu cho tiến trình.

Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể sẽ được hệ điều hành trực tiếp cấp phát tài nguyên hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu. Khi một tiến trình tạo tiến trình mới, tiến trình ban đầu có thể xử lý theo một trong hai khả năng sau; tiến trình cha tiếp tục xử lý đồng hành với tiến trình con. Tiến trình cha chờ đến khi một tiến trình con nào đó hoặc tất cả các tiến trình con kết thúc xử lý. Các hệ điều hành khác nhau có thể lựa chọn các cài đặt khác nhau để thực hiện thao tác tạo lập một 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 tương ứng của nó 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.

2.2. Luồng

Trong hầu hết các hệ điều hành, mỗi tiến trình có một không gian địa chỉ và chỉ có một dòng xử lý. Tuy nhiên có nhiều tình huống người sử dụng mong muốn có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ, và 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).

2.2.1. Định nghĩa

Một tiểu trình (threads) là một đơn vị xử lý cơ bản trong hệ thống.

Mỗi 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 tiến trình tạo thành những thực thể độc lập. Mỗi tiến trình có một tập tài nguyên và một môi trường riêng. 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. Ngược lại, 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.

2.2.2. Thông tin so sánh sự giống nhau và khác nhau của tiến trình và luồng

Trong lĩnh vực hệ điều hành, tiến trình và luồng là hai khái niệm cơ bản nhưng khác nhau. Chúng đều đóng vai trò quan trọng trong việc thực thi và quản lý các chương trình.

Bảng 2.1. Sự giống nhau của tiến trình và luồng

Các tiêu chí so sánh	Nội dung
Đơn vị thực thi	Cả tiến trình và luồng đều là các đơn vị cơ bản mà hệ điều hành sử dụng để quản lý thực thi chương trình.
Chạy đồng thời	Cả hai đều có thể chạy đồng thời, cho phép máy tính thực hiện nhiều tác vụ cùng lúc.
Sử dụng tài nguyên hệ thống	Cả tiến trình và luồng đều sử dụng các tài nguyên của hệ thống như CPU và bộ nhớ.
Lập lịch và quản lý	Hệ điều hành đảm nhận việc lập lịch và quản lý cả tiến trình lẫn luồng.

Bảng 2.2. Sự khác nhau của tiến trình và luồng

Các tiêu chí so sánh	Tiến trình (process)	Luồng (thread)
Định nghĩa và cấu trúc	Là một chương trình đang chạy, có không gian bộ nhớ riêng biệt.	Là một đơn vị nhỏ nhất của thực thi, nằm trong một tiến

	Mỗi tiến trình cung cấp môi trường chạy độc lập cho chương trình hoặc tập hợp các chương trình.	trình. Các luồng trong cùng một tiến trình có thể chia sẻ không gian bộ nhớ và tài nguyên.
Tài nguyên và không gian bộ nhớ	Có không gian bộ nhớ riêng biệt; tài nguyên không được chia sẻ giữa các tiến trình mặc định.	Các luồng trong cùng một tiến trình chia sẻ không gian bộ nhớ và tài nguyên, như heap, dữ liệu toàn cục,…
Chi phí tạo và quản lý	Việc tạo và quản lý tiến trình thường tốn kém hơn vì cần phân bổ không gian bộ nhớ riêng và tài nguyên hệ thống.	Tạo và quản lý luồng ít tốn kém hơn so với tiến trình vì chúng chia sẻ tài nguyên với tiến trình mẹ.
Hiệu suất	Chuyển đổi giữa các tiến trình có thể tốn nhiều thời gian hơn do cần lưu và phục hồi nhiều thông tin hơn.	Chuyển đổi giữa các luồng thường nhanh hơn vì chúng chia sẻ một số thông tin.
Độc lập	Mỗi tiến trình hoạt động độc lập và không phụ thuộc trực tiếp vào tiến trình khác.	Luồng phụ thuộc vào tiến trình mà chúng thuộc về; sự cố trong một luồng có thể ảnh hưởng đến các luồng khác trong cùng một tiến trình.
Truyền thông và tương tác	Truyền thông giữa các tiến trình thường phức tạp hơn và cần các kỹ thuật IPC (Inter-Process Communication).	Luồng có thể dễ dàng giao tiếp và chia sẻ dữ liệu với nhau do chung không gian bộ nhớ.

Hiểu biết về sự giống và khác nhau giữa tiến trình và luồng trong lập trình đa tiến trình và đa luồng, giúp tối ưu hóa hiệu suất và độ tin cậy của ứng dụng.

2.3. Điều phối tiến trình

2.3.1. Mục tiêu điều phối

Mục tiêu của việc điều phối: sự công bằng (fairness), tính hiệu quả (efficiency), thời gian đáp ứng hợp lý (response time), thời gian lưu lại trong hệ thống (turnaround), thông lượng tối đa (throughput). Bản thân các mục tiêu trên có thể có sự mâu thuẩn với nhau, nên chỉ có thể dung hòa theo một mức độ nào đó.

2.3.2. Các đặc điểm của tiến trình

Điều phối hoạt động của các tiến trình là một vấn đề rất phức tạp, đòi hỏi hệ điều hành khi giải quyết phải xem xét nhiều yếu tố khác nhau để có thể đạt được những mục tiêu đề ra. Một số đặc tính của tiến trình cần được quan tâm như sau: tính hướng xuất, nhập của tiến trình, tính hướng xử lý của tiến trình, tiến trình tương tác hay xử lý theo lô, độ ưu tiến của tiến trình, thời gian đã sử dụng CPU của tiến trình, thời gian còn lại tiến trình cần để hoàn tất.

2.3.3. Điều phối độc quyền và điều phối không độc quyền

Thuật toán điều phối cần xem xét và quyết định thời điểm chuyển đổi CPU giữa các tiến trình. Hệ điều hành có thể thực hiện cơ chế điều phối theo nguyên tắc độc quyền (preemptive) hoặc không độc quyền (non-preemptive).

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) hoặc khi tiến trình kết thúc. Các thuật toán điều phối độc quyền thường đơn giản và dễ cài đặt. Với các hệ thống tổng quát nhiều người dùng, thì chiến lược điều phối độc quyền thường không phù hợp,

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 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 hơn 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 (blocked), khi tiến trình chuyển từ trạng thái đang xử lý (running) sang trạng thái ready, khi tiến trình chuyển từ trạng thái chờ (blocked) sang trạng thái ready hoặc khi tiến trình kết thúc.

Các thuật toán điều phối theo nguyên tắc không độc quyền ngăn cản được tình trạng một tiến trình độc chiếm CPU, nhưng việc tạm dừng một tiến trình có thể dẫn đến các mâu thuẩn trong truy xuất, đòi hỏi phải sử dụng một phương pháp đồng bộ hóa thích hợp để giải quyết.

Trong các hệ thống sử dụng nguyên lý điều phối độc quyền có thể xảy ra tình trạng các tác vụ cần thời gian xử lý ngắn phải chờ tác vụ xử lý với thời gian rất dài hoàn tất. Nguyên lý điều phối độc quyền thường chỉ thích hợp với các hệ xử lý theo lô. Đối với các hệ thống tương tác (time sharing), các hệ thống thời gian thực (real time) cần phải sử dụng nguyên lý điều phối không độc quyền để các tiến trình quan trọng có cơ hội hồi đáp kịp thời. Tuy nhiên thực hiện điều phối theo nguyên lý không độc quyền đòi hỏi những cơ chế phức tạp trong việc phân định độ ưu tiên và phát sinh thêm chi phí khi chuyển đổi CPU qua lại giữa các tiến trình.

2.3.4. Độ ưu tiên của tiến trình

Để có cơ sở điều phối các tiến trình một cách hợp lý, hệ điều hành cần biết độ ưu tiên của từng tiến trình. Độ ưu tiên của một tiến trình là một giá trị giúp phân định tầm quan trọng của tiến trình. Nó có thể được phát sinh tự động bởi hệ thống hoặc được gán tường mình bởi người dùng. Độ ưu tiên có thể tĩnh hay động, và có thể được gán cho tiến trình một cách hợp lý hay ngẫu nhiên.

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 độ 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 cụ thể của hệ thống.

2.3.5. Tổ chức điều phối

Các loại danh sách sử dụng trong quá trình điều phối

Hệ điều hành sử dụng hai loại danh sách để thực hiện điều phối các tiến trình là danh sách sẵn sàng (ready list) và danh sách chờ (waiting list).

Khi một tiến trình bắt đầu đi vào hệ thống, nó được chèn vào danh sách các tác vụ (job list). Danh sách này bao gồm tất cả các tiến trình của hệ thống, các tiến trình đang thường trú trọng bộ nhớ chính và ở trạng thái sẵn sàng tiếp nhận CPU để hoạt động mới được đưa vào danh sách sẵn sàng.

Bộ điều phối sẽ chọn một tiến trình trong danh sách sẵn sàng và cấp CPU cho tiến trình đó. Tiến trình được cấp CPU sẽ thực hiện xử lý, và có thể chuyển qua trạng thái chờ khi xảy ra các sự kiện như đợi một thao tác nhập/xuất hoàn tất, yêu cầu tài nguyên chưa

được thỏa mãn, được yêu cầu tạm dừng,… khi đó tiến trình sẽ được chuyển sang một danh sách chờ đợi. Hệ điều hành chỉ sử dụng một danh sách sẵn sàng cho toàn hệ thống, nhưng mỗi một tài nguyên (thiết bị ngoại vi) có một danh sách chờ đợi riêng bao gồm các tiến trình đang chờ được cấp phát tài nguyên đó.

Quá trình xử lý của một tiến trình trãi qua những chu kỳ chuyển đổi qua lại giữa danh sách sẵn sàng và danh sách chờ đợi: Trước hết tiến trình mới được đưa vào danh sách sẵn sàng (ready list), nó sẽ đợi trong danh sách này cho đến khi được chọn để cấp phát CPU và bắt đầu xử lý. Sau đó có thể xảy ra một trong các tình huống sau: Thứ nhất, tiến trình phát sinh một yêu cầu tài nguyên mà hệ thống chưa thể đáp ứng, khi đó tiến trình sẽ được chuyển sang danh sách các tiến trình đang chờ tài nguyên tương ứng. Thứ hai, tiến trình có thể bị bắt buộc tạm dừng xử lý do một ngắt xảy ra, khi đó tiến trình được đưa trở lại vào danh sách sẵn sàng để chờ được cấp CPU cho lượt tiếp theo. Trong trường hợp đầu tiên, tiến trình cuối cùng sẽ chuyển từ trạng thái blocked sang trạng thái ready và lại đưa được trả vào danh sách sẵn sàng. Tiến trình lặp lại chu kỳ này cho đến khi hoàn tất tác vụ thì được hệ thống hủy bỏ khỏi mọi danh sách điều phối.

Các cấp độ điều phối

Việc điều phối được hệ điều hành thực hiện ở hai mức độ: điều phối tác vụ (job scheduling) và điều phối tiến trình (process scheduling).

Điều phối tác vụ

Quyết định lựa chọn tác vụ nào được đưa vào hệ thống, và nạp những tiến trình của tác vụ đó vào bộ nhớ chính để thực hiện. Chức năng điều phối tác vụ quyết định mức độ đa chương của hệ thống (số lượng tiến trình trong bộ nhớ chính). Khi hệ thống tạo lập một tiến trình, hay có một tiến trình kết thúc xử lý thì chức năng điều phối tác vụ mới được kích hoạt. Vì mức độ đa chương tương đối ổn định nên chức năng điều phối tác vụ có tần suất hoạt động thấp.

Để hệ thống hoạt động tốt, bộ điều phối tác vụ cần phân biệt tính chất của tiến trình là hướng nhập/xuất hay hướng xử lý. Một tiến trình được gọi là hướng nhập xuất nếu chủ yếu nó chỉ sử dụng CPU để thực hiện các thao tác nhập xuất. Ngược lại một tiến trình được gọi là hướng xử lý nếu chủ yếu nó chỉ sử dụng CPU để thực hiện các thao tác tính toán. Để cân bằng hoạt động của CPU và các thiết bị ngoại vi, bộ điều phối tác vụ nên

lựa chọn các tiến trình để nạp vào bộ nhớ sao cho hệ thống là sự pha trộn hợp lý giữa các tiến trình hướng nhập/xuất và các tiến trình hướng xử lý.

Điều phối tiến trình

Chọn một tiến trình ở trạng thái sẵn sàng (đã được nạp vào bộ nhớ chính và có đủ tài nguyên để hoạt động) và cấp phát CPU cho tiến trình đó thực hiện. Bộ điều phối tiến trình có tần suất hoạt động cao, sau mỗi lần xảy ra ngắt; thường là một lần trong khoảng 100ms. Do vậy để nâng cao hiệu suất của hệ thống, cần phải tăng tốc độ xử lý của bộ điều phối tiến trình. Chức năng điều phối tiến trình (điều phối CPU) là một trong những chức năng cơ bản và quan trọng nhất của hệ điều hành.

2.4. Các chiến lược điều phối tiến trình

2.4.1. Các tiêu chuẩn điều phối CPU

Mục này trình bày một số tiêu chuẩn cơ bản nhất: *thời gian đáp ứng, thời gian hoàn thành, thời gian chờ*; việc điều phối tiến trình nhằm làm cho các giá trị thời gian này là nhỏ nhất có thể.

Thời gian đáp ứng

Thời gian đáp ứng là khoảng thời gian tính từ thời điểm tiến trình được gửi vào hệ thống cho đến khi lần đầu tiên tiến trình đó được sử dụng CPU.

Thời gian hoàn thành

Thời gian hoàn thành là khoảng thời gian tính từ thời điểm tiến trình đến hệ thống cho đến khi tiến trình đó kết thúc; tức bằng thời gian kết thúc của tiến trình – thời gian đến.

Thời gian chờ đợi

Thời gian chờ là tổng thời gian chờ trong hàng đợi sẵn sàng; tức bằng thời gian hoàn thành – thời gian sử dụng CPU.

Lưu ý rằng các thuật toán điều phối CPU không ảnh hưởng đến các tiến trình đang thực hiện hay đang đợi thiết bị nhập xuất. Thời gian chờ có thể là thời gian để đưa tiến trình vào bộ nhớ, thời gian chờ trong hàng đợi sẵn sàng, thời gian chờ trong hàng đợi thiết bị.

Ví dụ 2.1

Cho 3 tiến trình P_1, P_2, P_3 có thời điểm vào hệ thống lần lượt là 0, 1, 2 và có thời gian xử lý lần lượt là 24, 3, 3. Hãy tính thời gian đáp ứng trung bình, thời gian hoàn thành trung bình và thời gian chờ trung bình của các tiến trình.

Hướng dẫn:

Biểu đồ Gantt (thứ tự cấp phát CPU) của các tiến trình là:

P_1	P_2	P_3	
0	24	27	30

Thời gian đáp ứng của từng tiến trình:

P_1: 0 – 0 =0

P_2: 24 – 1 =23

P_3: 27- 2= 25

Suy ra thời gian đáp ứng trung bình của các tiến trình là (0+23+25)/3=48/3=16 (ms).

Thời gian hoàn thành của từng tiến trình:

P_1: 24-0=24

P_2: 27-1=26

P_3: 30-2=28

Suy ra thời gian hoàn thành trung bình của các tiến trình là (24+26+28)/3=78/3=26 (ms).

Thời gian chờ của từng tiến trình:

P_1: 24-24=0

P_2: 26-3=23

P_3: 28-3=25

Suy ra thời gian chờ trung bình của các tiến trình là (0+23+25)/3=48/3=16 (ms).

2.4.2. Chiến lược điều phối "*đến trước phục vụ trước*"

Nguyên tắc:

Chiến lược điều phối "*đến trước phục vụ trước*" (first come first served - FCFS).

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 chiến lược 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 hoặc khi có một yêu cầu nhập/xuất.

Ví dụ 2.2

Process	Arrival time	Service time
P_1	0	24
P_2	1	3
P_3	2	3

Thông tin từ arrival time cho thấy thứ tự vào của các tiến trình là P_1, P_2, P_3.

Biểu đồ Gantt của các tiến trình là:

P_1	P_2	P_3	
0	24	27	30

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0
- Thời gian chờ của tiến trình P_2 là 23
- Thời gian chờ của tiến trình P_3 là 25

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (0 + 23 + 25)/3 =16 (ms).

Với chiến lược điều phối FCFS, các tiến trình có thời gian thực hiện ngắn phải chờ đợi như các tiến trình có thời gian thực hiện dài; do vậy thời gian chờ trung bình không tối ưu và thời gian chờ trung bình có sự thay đổi đáng kể khi thay đổi thứ tự thực hiện các tiến trình.

Xem lại **ví dụ 2.2** trên; trong đó có thay đổi thứ tự thực hiện các tiến trình.

Giả sử thứ tự vào của các tiến trình là P_2, P_3, P_1 như bảng sau:

Process	Arrival time	Service time
P_1	2	24
P_2	0	3
P_3	1	3

Biểu đồ Gantt của các tiến trình là:

P_2	P_3	P_1	
0	3	6	30

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 4
- Thời gian chờ của tiến trình P_2 là 0
- Thời gian chờ của tiến trình P_3 là 2

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (4 + 0 + 2)/3 = 2.67 (ms).

Chiến lược điều phối 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.

2.4.3. Chiến lược điều phối "xoay vòng"

Nguyên tắc: 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 chiến lược điều phối theo nguyên tắc không độc quyền: Khi một tiến trình sử dụng CPU đến hết thời gian quantum dành cho nó, hệ điều hành sẽ 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 2.3

Process	Arrival time	Service time
P_1	0	24
P_2	1	3
P_3	2	3

Nếu sử dụng *quantum time*=4ms, thứ tự cấp phát CPU là:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1	
0	4	7	10	14	18	22	26	30

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0+6=6
- Thời gian chờ của tiến trình P_2 là 3
- Thời gian chờ của tiến trình P_3 là 5

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (6 + 3 + 5)/3 = 4.67 (ms).

Nếu 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. Nếu quantum quá lớn sẽ làm tăng thời gian hồi đáp và giảm khả năng tương tác của hệ thống.

2.4.4. Chiến lược điều phối theo "độ ưu tiên"

Nguyên tắc:

Mỗi tiến trình gắn với 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. Độ ưu tiên cũng có thể được gán từ bên ngoài dựa vào các tiêu chuẩn do hệ điều hành quy định; chẳng hạn như tầm quan trọng của tiến trình, loại người sử dụng sở hữu tiến trình,…

Chiến lược 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 các tiến trình hiện đang xử lý. Chiến lược điều phối với độ ưu tiên 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. Chiến lược

điều phối với độ ưu tiên độc quyền sẽ chỉ đơn giản chèn tiến trình mới vào danh sách sẵn sàng, và tiến trình mới hiện hành vẫn tiếp tục xử lý hết thời gian dành cho nó.

Ví dụ 2.4

Process	Arrival time	Service time	Priority
P_1	0	24	3
P_2	1	3	1
P_3	2	3	2

Áp dụng chiến lược điều phối với độ ưu tiên theo nguyên tắc độc quyền

Biểu đồ Gantt của các tiến trình là:

P_1	P_2	P_3	
0	24	27	30

Thời gian chờ của tiến trình P_1 là 0

Thời gian chờ của tiến trình P_2 là 23

Thời gian chờ của tiến trình P_3 là 25

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (0 + 23 + 25)/3 = 16 (ms).

Kết quả này như chiến lược FCFS.

Áp dụng chiến lược điều phối với độ ưu tiên theo nguyên tắc không độc quyền

Biểu đồ Gantt cho lịch này là:

P_1	P_2	$P3$	P_1	
0	1	4	7	30

Thời gian chờ của tiến trình P_1 là 6

Thời gian chờ của tiến trình P_2 là 0

Thời gian chờ của tiến trình P_3 là 2

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (6 + 0 + 2)/3 = 2.67 (ms).

2.4.5. Chiến lược điều phối "*công việc ngắn nhất*"

Nguyên tắc:

Chiến lược điều phối "*công việc ngắn nhất*" (shortest job first - SJF) là một trường hợp đặc biệt của chiến lược điều phối với độ ưu tiên. Tiến trình có thời gian sử dụng CPU ít nhất sẽ sở hữu CPU. Chiến lược điều phối "*công việc ngắn nhất*" cũng có thể độc quyền (SJF) hoặc không độc quyền (shortest remaining time first - SRTF).

Ví dụ 2.5

Process	Arrival time	Service time
P_1	0	6
P_2	1	8
P_3	2	4
P_4	3	2

Áp dụng chiến lược SJF điều phối theo nguyên tắc độc quyền

Biểu đồ Gantt cho lịch này là:

P_1	P_4	P_3	P_2	
0	6	8	12	20

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0
- Thời gian chờ của tiến trình P_2 là 11
- Thời gian chờ của tiến trình P_3 là 6
- Thời gian chờ của tiến trình P_4 là 3

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (0 + 11 + 6 + 3)/4 = 5 (ms).

Áp dụng chiến lược SJF điều phối theo nguyên tắc không độc quyền

Biểu đồ Gantt cho lịch này là:

P_1	P_4	P_1	P_3	P_2	
0	3	5	8	12	20

(Lưu ý tại thời điểm 2, thời gian sử dụng CPU còn lại của P_1 và P_3 đều bằng 4, nên P_1 vẫn được cho thực hiện tiếp đến thời điểm 3).

Tính thời gian chờ của các tiến trình:

- Thời gian chờ của tiến trình P_1 là 0 + 2 = 2
- Thời gian chờ của tiến trình P_2 là 11

- Thời gian chờ của tiến trình P_3 là 6
- Thời gian chờ của tiến trình P_4 là 0

Thời gian chờ trung bình (average waiting time) của các tiến trình này là (2 + 11 + 6 + 0)/4 = 4.75 (ms).

2.5. Tắc nghẽn

2.5.1. Định nghĩa

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 mới có thể phát sinh được.

2.5.2. Điều kiện xảy ra tắc nghẽn

Các điều kiện cần có thể làm xuất hiện tắc nghẽn:

- Điều kiện 1: Có sử dụng tài nguyên không thể chia sẻ
- Điều kiện 2: Sự chiếm giữ và yêu cầu thêm tài nguyên
- Điều kiện 3: Không thể thu hồi tài nguyên từ tiến trình đang giữ chúng.
- Điều kiện 4: Tồn tại một chu trình 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 này thì không có tắc nghẽn.

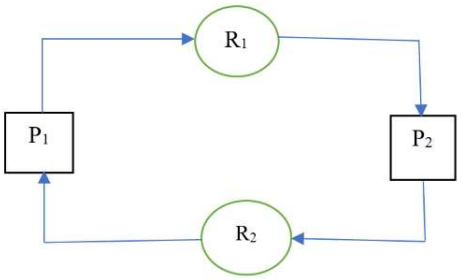
2.5.3. Đồ thị cấp phát tài nguyên

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.

P đang giữ R

P yêu cầu R

Sau đây là một ví dụ về một tình huống tắc nghẽn



Đồ thị này gọi là đồ thị cấp phát tài nguyên (resource allocation graph)

2.5.4. Các phương pháp xử lý tắc nghẽn

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

- Sử dụng một nghi thức (protocol) để đảm bảo 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.

2.5.5. Trạng thái an toàn

Tránh tắc nghẽn

Ngăn cản tắc nghẽn là mối bận tâm 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. Hệ điều hành sử dụng những cơ chế phức tạp để giải quyết vấn đề này.

Chuỗi cấp phát an toàn (safe sequence)

Một thứ tự các tiến trình <P_1, P_2,…,P_n> là an toàn (hay còn được gọi là **Chuỗi cấp phát an toàn**) đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình P_i, 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 P_j khác, với j<i.

Trạng thái an toàn (safe)

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

Trạng thái không an toàn (unsafe)`

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

Chiến lược cấp phát:

Chỉ thỏa mãn tài nguyên yêu cầu của tiến trình khi trạng thái kết thúc là an toàn.

2.5.6. Thuật toán Banker

Giải thuật xác định trạng thái an toàn

Cần sử dụng các cấu trúc dữ liệu sau:

int available[numresources]

available[r]: số lượng các thể hiện còn tự do của tài nguyên r.

int max[numprocs, numresources]

max[p,r]: nhu cầu tối đa của tiến trình p đối với tài nguyên r.

int allocation[numprocs, numresources]

allocation[p,r]:số tài nguyên r thực sự cấp phát cho tiến trình p.

int need[numprocs, numresources]

need[p,r] = max[p,r]- allocation[p,r];

Giải thuật xác định trạng thái an toàn

Bước 1:

Giả sử có các mảng:

int work [numprocs, numresources]= available;

int finish[numprocs]=false;

Bước 2:

Tìm i sao cho:

finish[i]==false và need[i] <= work[i]

Nếu không tìm thấy i thỏa mãn thì chuyển đến bước 4;

Bước 3:

work= work+ allocation[i];

finish[i]=true;

Chuyển đến bước 2;

Bước 4:

Nếu finish[i]==true với mọi *i* thì hệ thống ở trạng thái an toàn.

Ví dụ 2.6

Hệ thống gồm 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên R_1, $R2$, R_3; trong đó R_1 có 10 thể hiện, R_2 có 5 thể hiện, R_3 có 7 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 được mô tả như sau:

Process	Max			Allocation			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	7	5	3	0	1	0	3	3	2
P_1	3	2	2	2	0	0			
P_2	9	0	2	3	0	2			
P_3	2	2	2	2	1	1			
P_4	4	3	3	0	0	2			

Tại thời điểm T_0 hệ thống có ở trạng thái an toàn hay không ?

Process	Need		
	R_1	R_2	R_3
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Process	Work		
	R_1	R_2	R_3
	3	3	2
P_1	5	3	2
P_3	7	4	3
P_4	7	4	5
P_0	7	5	5
P_2	10	5	7

Hệ thống trên ở trạng thái an toàn vì nó tồn tại thứ tự an toàn P_1,P_3,P_4,P_0,P_2.

Giải thuật phát hiện tắc nghẽn

Cần sử dụng các cấu trúc dữ liệu sau:

int available[numresources]

available[r]: số lượng các thể hiện còn tự do của tài nguyên r.

int allocation[numprocs, numresources]

allocation[p,r]: số tài nguyên r thực sự cấp phát cho tiến trình p.

int request[numprocs, numresources]

request[p,r]: số tài nguyên r tiến trình p yêu cầu thêm.

Giải thuật phát hiện tắc nghẽn

Bước 1:

```
int work [numresources]=available;
int finish[numprocs];
for (i=0;i<numprocs;i++)
        finish[i]=(allocation[i]==0);
```

Bước 2:

Tìm *i* sao cho finish[i]==false và request[i]<=work;

Nếu không có i thỏa mãn, chuyển đến bước 4;

Bước 3:

```
work= work+allocation[i];
finish[i]=true;
```

Chuyển đến bước 2;

Bước 4:

Nếu finish[i]==true với mọi *i* thì hệ thống không có tắc nghẽn;

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

Ví dụ 2.7

Cho một hệ thống gồm 5 tiến trình P_0,P_1,P_2,P_3,P_4 và 3 loại tài nguyên R_1,R_2,R_3; trong đó R_1 có 7 thể hiện, R_2 có 2 thể hiện, R_3 có 6 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 được mô tả như sau:

	Request			Allocation			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	0	0	0	1	0	0	0	0
P_1	2	0	2	2	0	0			
P_2	0	0	0	3	0	3			
P_3	1	0	0	2	1	1			
P_4	0	0	2	0	0	2			

Tại thời điểm T_0 hệ thống có ở trạng thái an toàn hay không ?

Process	Work		
	R_1	R_2	R_3
	0	0	0
P_0	0	1	0
P_2	3	1	3
P_3	5	2	4
P_4	5	2	6
P_1	7	2	6

Vậy hệ thống không bị deadlock (hệ thống an toàn) vì tồn tại một thứ tự an toàn P_0,P_2,P_3,P_4,P_1.

Giải thuật yêu cầu tài nguyên

Giải thuật yêu cầu tài nguyên

Giả sử tiến trình P_i yêu cầu k thể hiện của tài nguyên r.

Bước 1:

Nếu k<=need[i], chuyển đến bước 2;

ngược lại, xảy ra tình huống lỗi;

Bước 2:

Nếu k<=available [i], chuyển đến bước 3;

ngược lại, P_i phải chờ;

Bước 3:

Giả sử hệ thống đã cấp phát cho P_i các tài nguyên mà nó yêu cầu và cập nhật tình trạng hệ thống như sau:

available [i]= available [i]-k;

allocation[i[=allocation[i[+k;

need[i]= need[i]-k;

Nếu trạng thái kết thúc là an toàn, lúc này các tài nguyên trên sẽ được cấp phát thật sự cho P_i; ngược lại, P_i phải chờ.

Ví dụ 2.8

Hệ thống gồm 5 tiến trình P_0, P_1, P_2, P_3, P_4 và 3 loại tài nguyên R_1, R_2, R_3; trong đó R_1 có 10 thể hiện, R_2 có 5 thể hiện, R_3 có 7 thể hiện.

Giả sử trạng thái của hệ thống tại thời điểm T_0 được mô tả như sau:

Process	Max			Allocation			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	7	5	3	0	1	0	3	3	2
P_1	3	2	2	2	0	0			
P_2	9	0	2	3	0	2			
P_3	2	2	2	2	1	1			
P_4	4	3	3	0	0	2			

Giả sử P_1 yêu cầu cấp phát với số thể hiện (1, 0, 2). Hỏi yêu cầu này có được thỏa mãn hay không ?

Tương tự với P_4(3, 3, 0), P_0(0, 2, 0), P_3(0, 2, 1).

Tiến trình P_1 yêu cầu thêm (1,0,2) thể hiện, ta cần cập nhật lại trạng thái allocation của P_1 là (3,0,2).

Kiểm tra các điều kiện:

Request[1](1,0,2) <= Available (3,3,2) - >True

Request[1](1,0,2) <= Need[1] (1,2,2) - >True

(Lưu ý: Nếu một trong hai điều kiện trên là sai thì kết luận việc yêu cầu cấp phát thêm không thành công; nếu cả hai điều kiện đều đúng thì mới tiến hành tính các ma trận Need và Work).

Tính ma trận Need: giữ nguyên giá trị cho các tiến trình P_0, P_2, P_3, P_4; chỉ cập nhật thông tin của P_1.

Process	Need		
	R_1	R_2	R_3
P_0	7	4	3
P_1	0	2	0
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Process	Work		
	R_1	R_2	R_3
	2	3	0
P_1	5	3	2
P_3	7	4	3
P_4	7	4	5
P_0	7	5	5
P_2	10	5	7

Tồn tại chuỗi an toàn P_1,P_3,P_4,P_0,P_2. Vậy yêu cầu cấp phát thêm của P_1 là thành công.