CHƯƠNG 3. TIẾN TRÌNH

I. Giới thiệu

Những hệ thống máy tính ban đầu cho phép chỉ một chương trình được thực thi tại một thời điểm. Chương trình này có toàn quyền điều khiển hệ thống và có truy xuất tới tất cả tài nguyên của hệ thống. Những hệ thống máy tính hiện nay cho phép nhiều chương trình được nạp vào bộ nhớ và được thực thi đồng hành. Sự phát triển này yêu cầu sự điều khiển mạnh mẽ hơn và phân chia nhiều hơn giữa các tiến trình.

Yêu cầu này dẫn đến khái niệm tiến trình, một chương trình đang thực thi. Tiến trình là một đơn vị công việc trong một hệ điều hành chia thời hiện đại. Một hệ điều hành phức tạp hơn được mong đợi nhiều hơn trong việc thực hiện các hành vi của người dùng. Mặc dù quan tâm chủ yếu của hệ điều hành là thực thi chương trình người dùng, nhưng nó cũng quan tâm đến các tác vụ khác nhau bên ngoài nhân. Do đó, một hệ thống chứa tập hợp các tiến trình: tiến trình hệ điều hành thực thi mã hệ thống, tiến trình người dùng thực thi mã người dùng. Tất cả tiến trình này có tiềm năng thực thi đồng hành, với một CPU (hay nhiều CPU) được đa hợp giữa chúng. Bằng cách chuyển đổi CPU giữa các tiến trình, hệ điều hành có thể làm cho máy tính hoạt động với năng suất cao hơn.

II. Khái niệm tiến trình

Một vấn đề cần thảo luận là cái gì được nhắc đến trong tất cả hoạt động của CPU? Một hệ thống theo lô thực thi công việc, trái lại một hệ thống chia sẻ thời gian thực thi chương trình người dùng hay tác vụ. Thậm chí trên hệ thống đơn người dùng như Microsoft Windows và Macintosh OS, một người dùng có thể chạy nhiều chương trình tại một thời điểm: bộ xử lý văn bản, trình duyệt web, e-mail. Thậm chí nếu người dùng có thể thực thi chỉ một tiến trình tại một thời điểm, thì một hệ điều hành cần hỗ trợ những hoạt động được lập trình bên trong, như quản lý bộ nhớ. Trong nhiều khía cạnh, tất cả hoạt động là tương tự vì thế chúng ta gọi tất cả chúng là tiến trình.

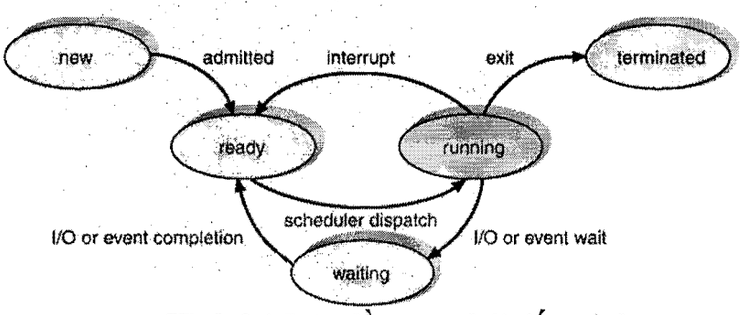
II.1 Tiến trình

Thật vậy, một tiến trình là một chương trình đang thực thi. Một tiến trình không chỉ là mà chương trình, nó còn bao gồm hoạt động hiện hành như được hiện diện bởi giá trị của bộ đếm chương trình và nội dung các thanh ghi của bộ xử lý. Ngoài ra, một tiến trình thường chứa ngăn xếp tiến trình, chứa dữ liệu tạm thời (như các tham số phương thức, các địa chỉ trả về, các biến cục bộ) và phần dữ liệu chứa các biến toàn cục.

Chúng ta nhấn mạnh rằng, một chương trình không phải là một tiến trình; một chương trình là một thực thể thụ động, như nội dung của các tập tin được lưu trên đĩa, trái lại một tiến trình là một thực thể chủ động, với một bộ đếm chương trình xác định chỉ thị lệnh tiếp theo sẽ thực thi và tập hợp tài nguyên có liên quan.

Mặc dù hai tiến trình có thể được liên kết với cùng chương trình nhưng chúng được chứa hai thứ tự thực thi riêng rẻ. Thí dụ, nhiều người dùng có thể đang chạy các bản sao của chương trình gửi nhận thư, hay cùng người dùng có thể nạp lên nhiều bản sao của một chương trình soạn thảo văn bản. Mỗi bản sao của chúng là một tiến trình riêng và mặc dù các phần văn bản là giống nhau, các phần dữ liệu khác nhau. Ngoài ra,

một tiến trình có thể tạo ra nhiều tiến trình khi nó thực thi.



Hình 3.1-Lưu đồ trạng thái tiến trình

II.2 Trạng thái tiến trình

Khi một tiến trình thực thi, nó thay đổi trạng thái. Trạng thái của tiến trình được định nghĩa bởi các hoạt động hiện hành của tiến trình đó. Mỗi tiến trình có thể ở một trong những trạng thái sau:

- Mới (new): tiến trình đang được tạo ra
- Đang chạy (running): các chỉ thị đang được thực thi
- Chờ (waiting): tiến trình đang chờ sự kiện xảy ra (như hoàn thành việc nhập/xuất hay nhận tín hiệu)
- Sẵn sàng (ready): tiến trình đang chờ được gán tới một bộ xử lý.
- Kết thúc (terminated): tiến trình hoàn thành việc thực thi

Các tên trạng thái này là bất kỳ, và chúng khác nhau ở các hệ điều hành khác nhau. Tuy nhiên, các trạng thái mà chúng hiện diện được tìm thấy trên tất cả hệ thống. Các hệ điều hành xác định mô tả trạng thái tiến trình. Chỉ một tiến trình có thể đang chạy tức thì trên bất kỳ bộ xử lý nào mặc dù nhiều tiến trình có thể ở trạng thái sẵn sàng và chờ.

II.3 Khối điều khiển tiến trình

Mỗi tiến trình được hiện diện trong hệ điều hành bởi một **khối điều khiển quá trình** (Process Control Block-PCB) – cũng được gọi khối điều khiển tác vụ. Một PCB được hiển thị trong hình 3.2. Nó chứa nhiều phần thông tin được gắn liền với một tiến trình xác định, gồm:

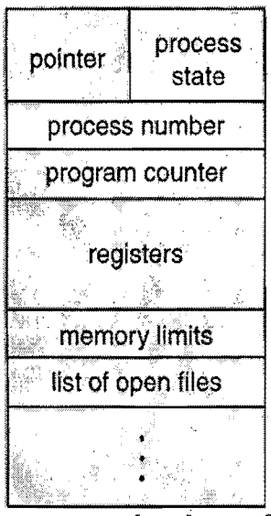
Trạng thái tiến trình (process state): trạng thái có thể là mới, sẵn sàng, đang chạy, chờ đợi, kết thúc, ...

Bộ đếm chương trình (program counter): bộ đếm hiển thị địa chỉ của chỉ thị kế tiếp được thực thi cho tiến trình này.

Các thanh ghi (registers) CPU: các thanh ghi khác nhau về số lượng và loại, phụ thuộc vào kiến trúc máy tính. Chúng gồm các bộ tổng (accumulators), các thanh ghi chỉ mục, các con trỏ ngăn xếp, và các thanh ghi đa năng (general- purpose registers), cùng với thông tin mã điều kiện (condition-code information). Cùng với bộ đếm chương trình, thông tin trạng thái này phải được lưu khi một ngắt xảy ra, cho

phép tiến trình được tiếp tục một cách phù hợp sau đó (Hình 3.3).

Thông tin lập lịch biểu CPU (CPU-scheduling information): thông tin gồm độ ưu tiên của tiến trình, các con trỏ chỉ tới các hàng đợi lập lịch biểu, và bất kỳ tham số lập lịch biểu khác.



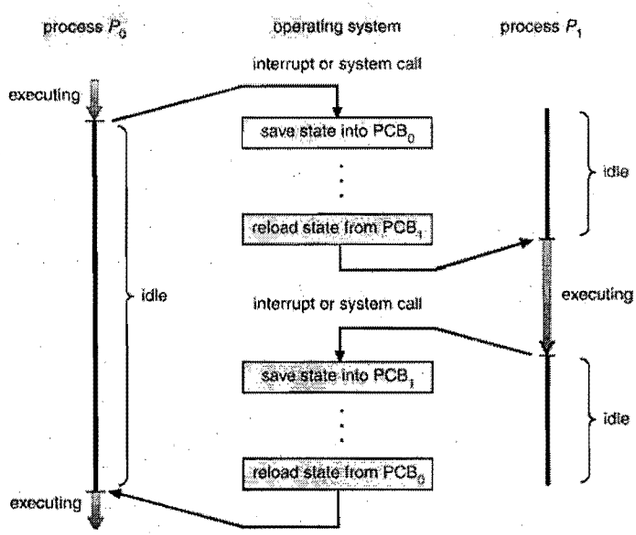
Thông tin quản lý bộ nhớ (Memory-management information): thông tin này có thể gồm những thông tin như giá trị của các thanh ghi nền và thanh ghi giới hạn, các bảng trang hay các bảng phân đoạn, phụ thuộc hệ thống bộ nhớ được dùng bởi hệ điều hành.

Thông tin tính toán (accounting information): thông tin này gồm lượng CPU và thời gian thực được dùng, công việc hay số tiến trình,…

Thông tin trạng thái nhập/xuất (I/O status information): thông tin này gồm danh sách của thiết bị nhập/xuất được cấp phát tiến trình này, một danh sách các tập tin đang mở,..

PCB đơn giản phục vụ như kho chứa cho bất cứ thông tin khác nhau từ tiến trình này tới tiến trình khác.

Hình 3.2-Khối điều khiển tiến trình



Hình 3.3-Lưu đồ hiển thị việc chuyển CPU từ tiến trình này tới tiến trình khác

II.4 Luồng

Mô hình tiến trình vừa được thảo luận ngụ ý rằng một tiến trình là một chương trình thực hiện một luồng đơn thực thi. Thí dụ, nếu một tiến trình đang chạy một chương trình xử lý văn bản, một luồng đơn của chỉ thị đang được thực thi. Đây là một luồng điều khiển đơn cho phép tiến trình thực thi chỉ một tác vụ tại một thời điểm. Thí dụ, người dùng không thể cùng lúc nhập các ký tự và chạy bộ kiểm tra chính tả trong cùng một tiến trình. Nhiều hệ điều hành hiện đại mở rộng khái niệm tiến trình để cho phép một tiến trình có nhiều luồng thực thi. Do đó, chúng cho phép thực hiện nhiều

hơn một tác vụ tại một thời điểm.

III. Lập lịch biểu tiến trình

Mục tiêu của việc đa chương là có vài tiến trình chạy tại tất cả thời điểm để tận dụng tối đa việc sử dụng CPU. Mục tiêu của chia thời là chuyển CPU giữa các quá trình thường xuyên để người dùng có thể giao tiếp với mỗi chương trình trong khi đang chạy. Một hệ thống đơn xử lý chỉ có thể chạy một tiến trình. Nếu nhiều hơn một tiến trình tồn tại, các tiến trình còn lại phải chờ cho tới khi CPU rảnh và có thể lập lịch biểu lại.

III.1 Hàng đợi lập lịch biểu

Khi các tiến trình được đưa vào hệ thống, chúng được đặt vào hàng đợi công việc. Hàng đợi chứa tất cả tiến trình trong hệ thống. Các tiến trình đang nằm trong bộ nhớ chính sẵn sàng và chờ để thực thi được giữ trên một danh sách được gọi là hàng đợi sẵn sàng. Hàng đợi này thường được lưu như một danh sách liên kết. Đầu của hàng đợi sẵn sàng chứa hai con trỏ: một chỉ đến PCB đầu tiên và một chỉ tới PCB cuối cùng trong danh sách. Chúng ta bổ sung thêm trong mỗi PCB một trường con trỏ chỉ tới PCB kế tiếp trong hàng đợi sẵn sàng.

Hệ điều hành cũng có các hàng đợi khác. Khi một tiến trình được cấp phát CPU, nó thực thi một khoảng thời gian và cuối cùng kết thúc, được ngắt, hay chờ một sự kiện xác định xảy ra, chẳng hạn như hoàn thành một yêu cầu nhập/xuất. Trong trường hợp yêu cầu nhập/xuất, một yêu cầu có thể là ổ đĩa băng từ tận hiến hay một thiết bị được chia sẻ như đĩa. Vì hệ thống có nhiều tiến trình, đĩa có thể bận với yêu cầu nhập/xuất của các tiến trình khác. Do đó, tiến trình phải chờ đĩa. Danh sách quá trình chờ một thiết bị nhập/xuất cụ thể được gọi là hàng đợi thiết bị. Mỗi thiết bị có hàng đợi của chính nó như hình 3.4.

Một biểu diễn chung của lập lịch biểu tiến trình là một lưu đồ hàng đợi, như hình 3.5. Mỗi hình chữ nhật hiện diện một hàng đợi. Hai loại hàng đợi được hiện diện: hàng đợi sẵn sàng và tập các hàng đợi thiết bị, vòng tròn hiện diện tài nguyên phục vụ hàng đợi, các mũi tên hiển thị dòng các tiến trình trong hệ thống.

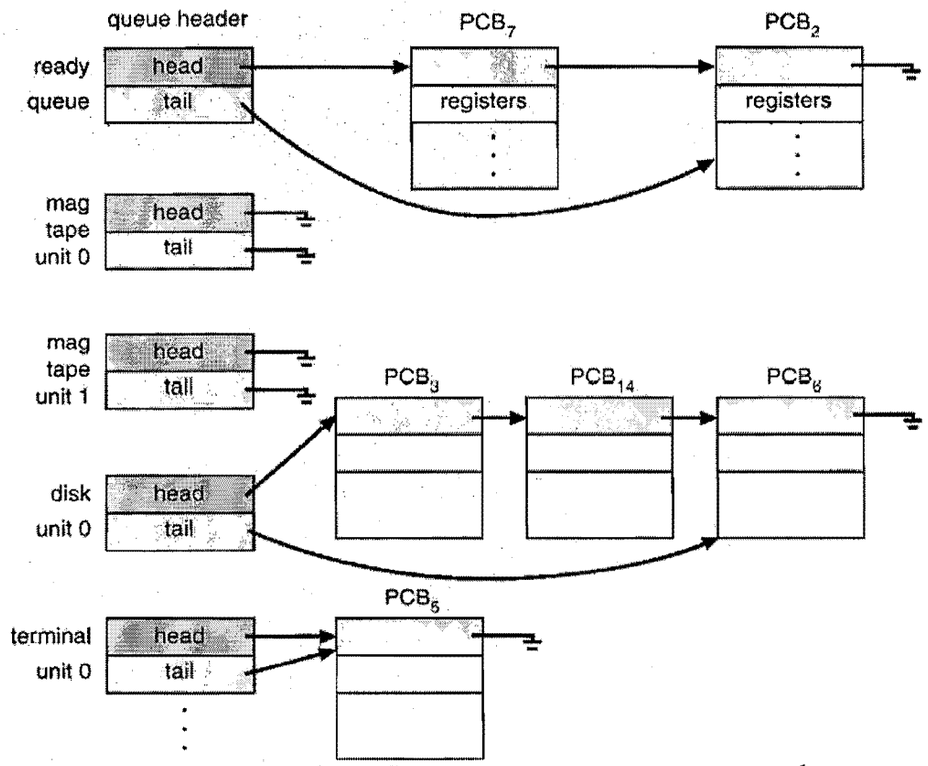
Một tiến trình mới khởi đầu được đặt vào hàng đợi. Nó chờ trong hàng đợi sẵn sàng cho tới khi nó được chọn thực thi. Một khi tiến trình được gán tới CPU và đang thực thi, một trong nhiều sự kiện có thể xảy ra:

- Tiến trình có thể phát ra một yêu cầu nhập/xuất và sau đó được đặt vào trong hàng đợi nhập/xuất.
- Tiến trình có thể tạo một tiến trình con và chờ cho tiến trình con kết thúc
- Khi một ngắt xảy ra, tiến trình có thể bị buộc trả lại CPU và được đặt trở lại trong hàng đợi sẵn sàng.

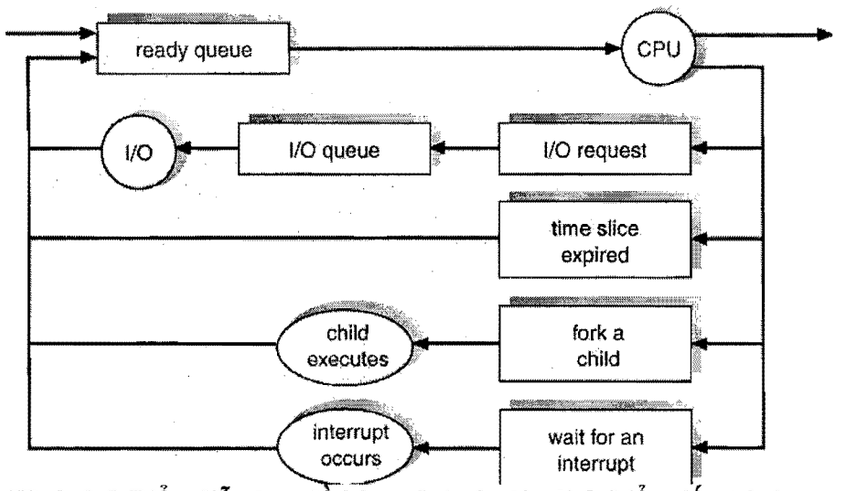
Trong hai trường hợp đầu, cuối cùng tiến trình chuyển từ trạng thái chờ tới trạng thái sẵn sàng và sau đó đặt trở lại vào hàng đợi sẵn sàng. Một tiến trình tiếp tục chu kỳ này cho tới khi nó kết thúc. Tại thời điểm kết thúc nó bị xoá từ tất cả hàng đợi. Sau đó, PCB và tài nguyên của nó được thu hồi.

III.2 Bộ lập lịch biểu

Một tiến trình di dời giữa hai hàng đợi lập lịch khác nhau suốt thời gian sống của nó. Hệ điều hành phải chọn, cho mục đích lập lịch, các tiến trình từ các hàng đợi này. Tiến trình chọn lựa này được thực hiện bởi bộ lập lịch hợp lý.



Hình 3.4-Hàng đợi sẵn sàng và các hàng đợi nhập/xuất khác nhau



Hình 3.5-Biểu diễn lưu đồ hàng đợi của lập lịch biểu tiến trình

Trong hệ thống theo lô, thường nhiều hơn một tiến trình được gửi đến hơn là có thể được thực thi tức thì. Các tiến trình này được lưu tới thiết bị lưu trữ (như đĩa), nơi chúng được giữ cho việc thực thi sau đó. **Bộ lập lịch dài** (long-term scheduler) hay bộ lập lịch công việc (job scheduler), chọn các tiến trình từ vùng đệm và nạp chúng vào bộ nhớ để thực thi. **Bộ lập lịch ngắn** (short-term scheduler) hay bộ lập lịch CPU chọn một tiến trình từ các tiến trình sẵn sàng thực thi và cấp phát CPU cho quá trình đó.

Sự khác biệt chủ yếu giữa hai bộ lập lịch là tính thường xuyên của việc thực thi. Bộ lập lịch CPU phải chọn một tiến trình mới cho CPU thường xuyên. Một quá trình có thể thực thi chỉ một vài mili giây trước khi chờ yêu cầu nhập/xuất. Bộ lập lịch CPU thường thực thi ít nhất một lần mỗi 100 mili giây. Vì thời gian ngắn giữa việc thực thi

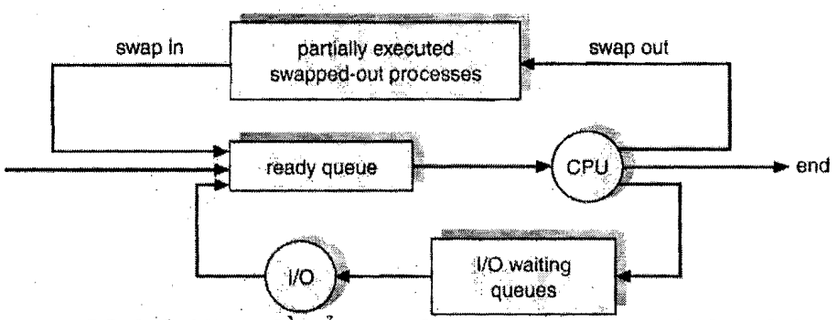
nên bộ lập lịch phải nhanh. Nếu nó mất 10 mili giây để quyết định thực thi một tiến trình 100 mili giây thì 10/(100+10) = 9 phần trăm của CPU đang được dùng (hay bị lãng phí) đơn giản cho lập lịch công việc.

Ngược lại, bộ lập lịch công việc thực thi ít thường xuyên hơn. Có vài phút giữa việc tạo các tiến trình mới trong hệ thống. Bộ lập lịch công việc điều khiển mức độ đa chương – số tiến trình trong bộ nhớ. Nếu mức độ đa chương ổn định thì tốc độ trung bình của việc tạo tiến trình phải bằng tốc độ khởi hành trung bình của tiến trình rời hệ thống. Vì khoảng thời gian dài hơn giữa việc thực thi nên bộ lập lịch công việc có thể cấp nhiều thời gian hơn để chọn một tiến trình thực thi.

Bộ lập lịch công việc phải thực hiện một chọn lựa cẩn thận. Thông thường, hầu hết các tiến trình có thể được mô tả như là **tiến trình hướng nhập/xuất** (I/O- bound proces) hay **tiến trình hướng CPU** (CPU-bound process). Một tiến trình hướng nhập/xuất mất nhiều thời gian hơn để thực hiện nhập/xuất hơn thời gian tính toán. Ngược lại, một tiến trình hướng CPU phát sinh các yêu cầu nhập/xuất không thường xuyên, dùng thời gian để thực hiện việc tính toán hơn một tiến trình hướng nhập/xuất dùng. Bộ lập lịch công việc nên chọn sự kết hợp hài hoà giữa tiến trình hướng nhập/xuất và tiến trình hướng CPU. Nếu tất cả tiến trình là hướng nhập/xuất thì hàng đợi sẵn sàng sẽ luôn rỗng và bộ lập lịch CPU sẽ có ít công việc để thực hiện.

Nếu tất cả tiến trình là hướng CPU thì hàng đợi nhập/xuất sẽ luôn rỗng, các thiết bị sẽ không được sử dụng và hệ thống sẽ mất cân bằng. Hệ thống với năng lực tốt nhất sẽ có sự kết hợp các tiến trình hướng CPU và hướng nhập/xuất.

Trên một vài hệ thống, bộ lập lịch công việc có thể không có hay rất ít. Thí dụ, các hệ thống chia thời như UNIX thường không có bộ lập lịch công việc nhưng đơn giản đặt mỗi tiến trình mới vào bộ nhớ cho bộ lập lịch CPU. Khả năng ổn định của hệ thống này phụ thuộc vào giới hạn vật lý (như số lượng thiết bị cuối sẵn dùng) hay tính tự nhiên tự chuyển đổi của người dùng. Nếu năng lực thực hiện giảm tới mức độ không thể chấp nhận được thì một số người dùng sẽ thoát khỏi hệ thống.



Hình 3.6-Lưu đồ bổ sung lập lịch trung gian tới hàng đợi

Một số hệ thống như hệ chia thời có thể đưa vào một cấp độ lập lịch bổ sung hay lập lịch trung gian. **Bộ lập lịch trung gian** (medium-term process) này (được hiển thị trong lưu đồ hình 3.6) xóa các tiến trình ra khỏi bộ nhớ (từ sự canh tranh CPU) và do đó giảm mức độ đa chương. Tại thời điểm sau đó, tiến trình có thể được đưa trở lại bộ nhớ và việc thực thi của nó có thể được tiếp tục nơi nó bị đưa ra. Cơ chế này được gọi là **hoán vị** (swapping). Tiến trình được hoán vị ra và sau đó được hoán vị vào bởi bộ lập lịch trung gian. Hoán vị là cần thiết để cải tiến sự trộn lẫn tiến trình (giữa các tiến trình hướng nhập/xuất và hướng CPU), hay vì một thay đổi trong yêu cầu bộ nhớ vượt quá kích thước bộ nhớ sẵn dùng. Hoán vị sẽ được thảo luận trong chương sau.

III.3 Chuyển ngữ cảnh

Chuyển CPU tới một tiến trình khác yêu cầu lưu trạng thái của tiến trình cũ và nạp trạng thái được lưu cho tiến trình mới. Tác vụ này được xem như **chuyển ngữ cảnh** (context switch). **Ngữ cảnh** của tiến trình được hiện diện trong PCB của quá trình; Nó chứa giá trị các thanh ghi, trạng thái tiến trình (hình 3.1) và thông tin quản lý bộ nhớ. Khi chuyển ngữ cảnh ngữ cảnh xảy ra, nhân lưu ngữ cảnh của tiến trình cũ trong PCB của nó và nạp ngữ cảnh được lưu của tiến trình mới được lập lịch để chạy. Thời gian chuyển ngữ cảnh là chi phí thuần vì hệ thống không thực hiện công việc có ích trong khi chuyển. Tốc độ của nó khác từ máy này tới máy khác phụ thuộc vào tốc độ bộ nhớ, số lượng thanh ghi phải được chép và sự tồn tại của các chỉ thị đặc biệt (như chỉ thị để nạp và lưu tất cả thanh ghi). Điển hình dãy tốc độ từ 1 tới 1000 mili giây.

Những lần chuyển đổi ngữ cảnh phụ thuộc nhiều vào hỗ trợ phần cứng. Thí dụ, vài bộ xử lý (như Sun UltraSPARC) cung cấp nhiều tập thanh ghi. Một chuyển ngữ cảnh đơn giản chứa chuyển đổi con trỏ tới tập thanh ghi hiện hành. Dĩ nhiên, nếu quá trình hoạt động vượt quá tập thanh ghi thì hệ thống sắp xếp lại dữ liệu thanh ghi tới và từ bộ nhớ. Cũng vì thế mà hệ điều hành phức tạp hơn và nhiều công việc được làm hơn trong khi chuyển ngữ cảnh. Kỹ thuật quản lý bộ nhớ nâng cao có thể yêu cầu dữ liệu bổ sung để được chuyển với mỗi ngữ cảnh. Thí dụ, không gian địa chỉ của quá trình hiện hành phải được lưu khi không gian của tác vụ kế tiếp được chuẩn bị dùng. Không gian địa chỉ được lưu như thế nào và lượng công việc được yêu cầu để lưu nó phụ thuộc vào phương pháp quản lý bộ nhớ của hệ điều hành. Chuyển ngữ cảnh có thể dẫn đến thắt cổ chai năng lực thực hiện vì thế các lập trình viên đang sử dụng các cấu trúc mới để tránh nó bất cứ khi nào có thể.

IV. Thao tác trên tiến trình

Các tiến trình trong hệ thống có thể thực thi đồng hành và chúng phải được tạo và xóa tự động. Do đó, hệ điều hành phải cung cấp một cơ chế (hay phương tiện) cho việc tạo tiến trình và kết thúc tiến trình.

IV.1. Tạo tiến trình

Tiến trình có thể tạo nhiều tiến trình mới, bằng một lời gọi hệ thống **create-process,** trong khi thực thi. Tiến trình tạo gọi là tiến trình cha, ngược lại các tiến trình mới được gọi là tiến trình con của tiến trình đó. Mỗi tiến trình mới này sau đó có thể tạo các tiến trình khác, hình thành một cây tiến trình (hình 3.7).

Thông thường, một tiến trình sẽ cần các tài nguyên xác định (như thời gian CPU, bộ nhớ, tập tin, thiết bị nhập/xuất) để hoàn thành tác vụ của nó. Khi một quá trình tạo một tiến trình con, tiến trình con có thể nhận tài nguyên của nó trực tiếp từ hệ điều hành hay nó có thể bị ràng buộc tới một tập con các tài nguyên của tiến trình cha.

Tiến trình cha có thể phải phân chia các tài nguyên giữa các tiến trình con hay có thể chia sẻ một số tài nguyên (như bộ nhớ và tập tin) giữa nhiều tiến trình con. Giới hạn một tiến trình con tới một tập con tài nguyên của tiến trình cha ngăn chặn bất cứ quá trình từ nạp chồng hệ thống bằng cách tạo quá nhiều tiến trình con.

Ngoài ra, khi một tiến trình được tạo nó lấy tài nguyên vật lý và logic khác nhau, dữ liệu khởi tạo (hay nhập) có thể được truyền từ tiến trình cha tới tiến trình con. Thí dụ, xét một tiến trình với toàn bộ chức năng là hiển thị trạng thái của một tập tin F_1 trên màn hình. Khi nó được tạo, nó sẽ lấy dữ liệu vào từ tiến trình cha, tên của tập tin F_1 và nó sẽ thực thi dùng dữ liệu đó để lấy thông tin mong muốn. Nó có thể cũng lấy

tên của thiết bị xuất. Một số hệ điều hành chuyển tài nguyên tới tiến trình con. Trên hệ thống như thế, tiến trình mới có thể lấy hai tập tin đang mở, F_1 và thiết bị cuối, và có thể chỉ cần chuyển dữ liệu giữa hai tập tin. Khi một tiến trình tạo một tiến trình mới, hai khả năng có thể tồn tại trong thuật ngữ của việc thực thi:

- Tiến trình cha tiếp tục thực thi đồng hành với tiến trình con của nó.
- Tiến trình cha chờ cho tới khi một vài hay tất cả tiến trình con kết thúc.

Cũng có hai khả năng trong thuật ngữ không gian địa chỉ của tiến trình mới:

- Tiến trình con là bản sao của tiến trình cha.
- Tiến trình con có một chương trình được nạp vào nó.

Để hiển thị việc cài đặt khác nhau này, chúng ta xem xét hệ điều hành UNIX. Trong UNIX, mỗi tiến trình được xác định bởi **danh biểu tiến trình** (process identifier), là số nguyên duy nhất. Một tiến trình mới được tạo bởi lời gọi hệ thống **fork.** Tiến trình mới chứa bản sao của không gian địa chỉ của tiến trình gốc. Cơ chế này cho phép tiến trình cha giao tiếp dễ dàng với tiến trình con. Cả hai tiến trình (cha và con) tiếp tục thực thi tại chỉ thị sau khi lời gọi hệ thống **fork,** với một sự khác biệt: mã trả về cho lời gọi hệ thống fork là không cho tiến trình mới (con), ngược lại danh biểu tiến trình (khác không) của tiến trình con được trả về tới tiến trình cha. Điển hình lời gọi hệ thống **execlp** được dùng sau lời gọi hệ thống **fork** bởi một trong hai tiến trình để thay thế không gian bộ nhớ với tiến trình mới. Lời gọi hệ thống **execlp** nạp tập tin nhị phân vào trong bộ nhớ-xóa hình ảnh bộ nhớ của chương trình chứa lời gọi hệ thống **execlp** - và bắt đầu việc thực thi của nó. Trong cách thức này, hai tiến trình có thể giao tiếp và sau đó thực hiện cách riêng của nó. Sau đó, tiến trình cha có thể tạo nhiều hơn tiến trình con, hay nếu nó không làm gì trong thời gian quá trình con chạy thì nó sẽ phát ra lời gọi hệ thống **wait** để di chuyển nó vào hàng đợi sẵn sàng cho tới khi tiến trình con kết thúc. Chương trình C (hình 3.8 dưới đây) hiển thị lời gọi hệ thống UNIX được mô tả trước đó. Tiến trình cha tạo một tiến trình con sử dụng lời gọi hệ thống **fork.** Bây giờ chúng ta có hai tiến trình khác nhau chạy một bản sao của cùng chương trình. Giá trị **pid** cho tiến trình con là 0; cho tiến trình cha là một số nguyên lớn hơn 0. Tiến trình con phủ lắp không gian địa chỉ của nó với lệnh của UNIX là **/bin/ls** (được dùng để liệt kê thư mục) dùng lời gọi hệ thống **execlp.** Quá trình cha chờ cho tiến trình con hoàn thành với lời gọi hệ thống **wait.** Khi tiến trình con hoàn thành, tiến trình cha bắt đầu lại từ lời gọi hệ thống **wait** nơi nó hoàn thành việc sử dụng lời gọi hệ thống **exit.**

Ngược lại, hệ điều hành DEC VMS tạo một tiến trình mới, nạp chương trình xác định trong tiến trình đó và bắt đầu thực thi nó. Hệ điều hành Microsoft Windows NT hỗ trợ cả hai mô hình: không gian địa chỉ của tiến trình cha có thể được sao lại hay tiến trình cha có thể xác định tên của một chương trình cho hệ điều hành nạp vào không gian địa chỉ của tiến trình mới.

```
#include<stdio.h>
main(int argc, char* argv[])
{
Int pid;
        /*fork another process*/
    Pid = fork();
    if(pid<0){/*error occurred */
```

```
                fprintf(stderr, "Fork Failed");
                exit(-1);
}
else if (pid==0){/*child process*/
    execlp("/bin/ls","ls",NULL);
}
else { /*parent process*/
        /*parent will wait for the child to complete*/
        wait(NULL);
        printf("Child Complete");
        exit(0);
}
}
```

Hình 3.8-Chương trình C tạo một tiến trình riêng rẻ

IV.2. Kết thúc tiến trình

Một tiến trình kết thúc khi nó hoàn thành việc thực thi câu lệnh cuối cùng và yêu cầu hệ điều hành xóa nó bằng cách sử dụng lời gọi hệ thống **exit.** Tại thời điểm đó, tiến trình có thể trả về dữ liệu (đầu ra) tới tiến trình cha (bằng lời gọi hệ thống **wait).** Tất cả tài nguyên của tiến trình –gồm bộ nhớ vật lý và logic, các tập tin đang mở, vùng đệm nhập/xuất-được thu hồi bởi hệ điều hành. Việc kết thúc xảy ra trong các trường hợp khác. Một tiến trình có thể gây kết thúc một tiến trình khác bằng một lời gọi hệ thống hợp lý (thí dụ, abort). Thường chỉ có tiến trình cha bị kết thúc có thể gọi lời gọi hệ thống như thế. Ngược lại, người dùng có thể tùy ý giết (kill) mỗi công việc của tiến trình còn lại. Do đó, tiến trình cha cần biết các định danh của các tiến trình con. Vì thế khi một tiến trình tạo một tiến trình mới, định danh của mỗi tiến trình mới được tạo được truyền tới cho tiến trình cha.

Một tiến trình cha có thể kết thúc việc thực thi của một trong những tiến trình con với nhiều lý do khác nhau:

- Tiến trình con sử dụng tài nguyên vượt quá mức được cấp. Điều này yêu cầu tiến trình cha có một cơ chế để xem xét trạng thái của các tiến trình con.
- Công việc được gán tới tiến trình con không còn cần thiết nữa.
- Tiến trình cha đang kết thúc và hệ điều hành không cho phép một tiến trình con tiếp tục nếu tiến trình cha kết thúc. Trên những hệ thống như thế, nếu một tiến trình kết thúc (bình thường hoặc không bình thường), thì tất cả tiến trình con cũng phải kết thúc. Trường hợp này được xem như kết thúc xếp tầng (cascading termination) thường được khởi tạo bởi hệ điều hành.

Để hiển thị việc thực thi và kết thúc tiến trình, xem xét hệ điều hành UNIX chúng ta có thể kết thúc một tiến trình dùng lời gọi hệ thống **exit;** nếu tiến trình cha có thể chờ cho đến khi tiến trình con kết thúc bằng lời gọi hệ thống **wait.** Lời gọi hệ

thống **wait** trả về định danh của tiến trình con bị kết thúc để tiến trình cha có thể xác định những tiến trình con nào có thể kết thúc. Tuy nhiên, nếu tiến trình cha kết thúc thì tất cả tiến trình con của nó được gán như tiến trình cha mới của chúng, tiến trình khởi tạo (init process). Do đó, các tiến trình con chỉ có một tiến trình cha để tập hợp trạng thái và thống kê việc thực thi.

IV.3. Hợp tác tiến trình

Các tiến trình đồng hành thực thi trong hệ điều hành có thể là những tiến trình độc lập hay những tiến trình hợp tác. Một tiến trình là **độc lập** (independent) nếu nó không thể ảnh hưởng hay bị ảnh hưởng bởi các tiến trình khác thực thi trong hệ thống.

Rõ ràng, bất kỳ một tiến trình không chia sẻ bất cứ dữ liệu (tạm thời hay cố định) với tiến trình khác là độc lập. Ngược lại, một tiến trình là **hợp tác** (cooperating) nếu nó có thể ảnh hưởng hay bị ảnh hưởng bởi các tiến trình khác trong hệ thống. Hay nói cách khác, bất cứ tiến trình chia sẻ dữ liệu với tiến trình khác là tiến trình hợp tác. Chúng ta có thể cung cấp một môi trường cho phép hợp tác tiến trình với nhiều lý do:

- **Chia sẻ thông tin:** vì nhiều người dùng có thể quan tâm cùng phần thông tin (thí dụ, tập tin chia sẻ), chúng phải cung cấp một môi trường cho phép truy xuất đồng hành tới những loại tài nguyên này.

- **Gia tăng tốc độ tính toán:** nếu chúng ta muốn một tác vụ chạy nhanh hơn, chúng ta phải chia nó thành những tác vụ nhỏ hơn, mỗi tác vụ sẽ thực thi song song với các tác vụ khác. Việc tăng tốc như thế có thể đạt được chỉ nếu máy tính có nhiều thành phần đa xử lý (như các CPU hay các kênh I/O).

- **Tính module hóa:** chúng ta muốn xây dựng hệ thống trong một kiểu mẫu dạng module, chia các chức năng hệ thống thành những tiến trình hay luồng như đã thảo luận ở chương trước.

- **Tính tiện dụng:** Thậm chí một người dùng đơn có thể có nhiều tác vụ thực hiện tại cùng thời điểm. Thí dụ, một người dùng có thể đang soạn thảo, in, và biên dịch cùng một lúc.

Thực thi đồng hành của các tiến trình hợp tác yêu cầu các cơ chế cho phép các tiến trình giao tiếp với các tiến trình khác và đồng bộ hóa các hoạt động của chúng.

Để minh họa khái niệm của các tiến trình cộng tác, chúng ta xem xét bài toán người sản xuất-người tiêu thụ, là mô hình chung cho các tiến trình hợp tác. Tiến trình người sản xuất cung cấp thông tin được tiêu thụ bởi tiến trình người tiêu thụ. Thí dụ, một chương trình in sản xuất các ký tự được tiêu thụ bởi trình điều khiển máy in. Một trình biên dịch có thể sản xuất mã hợp ngữ được tiêu thụ bởi trình hợp ngữ. Sau đó, trình hợp ngữ có sản xuất module đối tượng, được tiêu thụ bởi bộ nạp.

Để cho phép người sản xuất và người tiêu thụ chạy đồng hành, chúng ta phải có sẵn một vùng đệm chứa các sản phẩm có thể được điền vào bởi người sản xuất và được lấy đi bởi người tiêu thụ. Người sản xuất có thể sản xuất một sản phẩm trong khi người tiêu thụ đang tiêu thụ một sản phẩm khác. Người sản xuất và người tiêu thụ phải được đồng bộ để mà người tiêu thụ không cố gắng tiêu thụ một sản phẩm mà chưa được sản xuất. Trong trường hợp này, người tiêu thụ phải chờ cho tới khi các sản phẩm mới được tạo ra.

Bài toán người sản xuất-người tiêu thụ với vùng đệm không bị giới hạn (unbounded-buffer) thiết lập không giới hạn kích thước của vùng đệm. Người tiêu thụ có thể phải chờ các sản phẩm mới nhưng người sản xuất có thể luôn tạo ra sản phẩm

mới. Vấn đề người sản xuất-người tiêu thụ với vùng đệm có kích thước giới hạn (bounded-buffer) đảm bảo một kích thước cố định cho vùng đệm. Trong trường hợp này, người tiêu thụ phải chờ nếu vùng đệm rỗng, và người sản xuất phải chờ nếu vùng đệm đầy.

Vùng đệm có thể được cung cấp bởi hệ điều hành thông qua việc sử dụng phương tiện **giao tiếp liên tiến trình** (interprocess-communication-IPC), hay được mã hóa cụ thể bởi người lập trình ứng dụng với việc sử dụng bộ nhớ được chia sẻ. Để chúng ta hiển thị một giải pháp chia sẻ bộ nhớ đối với vấn đề vùng đệm bị giới hạn (bounded-buffer). Tiến trình người sản xuất và người tiêu thụ chia sẻ các biến sau:

```
#define BUFFER_SIZE[10]
typedef struct{
        ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Vùng đệm được chia sẻ được cài đặt như một mảng vòng với hai con trỏ logic: **in** và **out.** Biến **in** chỉ tới vị trí trống kế tiếp trong vùng đệm; **out** chỉ tới vị trí đầy đầu tiên trong vùng đệm. Vùng đệm là rỗng khi **in==out;** vùng đệm là đầy khi **((in + 1)%BUFFER_SIZE) ==out.**

Mã cho tiến trình người sản xuất và người tiêu thụ được trình bày dưới đây. Tiến trình người sản xuất có một biến **nextProduced** trong đó sản phẩm mới được tạo ra và được lưu trữ:

```
while (1) {
        /*produce an item in nextProduced*/
        while (((in + 1)%BUFFER_SIZE) ==out)
                ; /*do nothing*/
        buffer[in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
}
```

Tiến trình người tiêu thụ có biến cục bộ **nextConsumed** trong đó sản phẩm được tiêu thụ và được lưu trữ:

```
while (1){
while (in==out); //nothing
        nextConsumed =buffer[out];
        out = (out  +1) % BUFFER_SIZE;
        /*consume the item in nextConsumed*/
        }
```

Cơ chế này cho phép nhiều nhất BUFFER_SIZE –1 trong vùng đệm tại cùng một thời điểm.

V. Giao tiếp liên tiến trình

Trong phần trên chúng ta đã hiển thị cách mà các tiến trình hợp tác có thể giao tiếp với nhau trong một môi trường chia sẻ bộ nhớ. Cơ chế yêu cầu các tiến trình này chia sẻ nhóm vùng đệm chung và mã cho việc cài đặt vùng đệm được viết trực tiếp bởi người lập trình ứng dụng. Một cách khác đạt được cùng ảnh hưởng cho hệ điều hành là cung cấp phương tiện cho các tiến trình hợp tác giao tiếp với nhau bằng một phương tiện giao tiếp liên tiến trình (IPC). IPC cung cấp một cơ chế cho phép một tiến trình giao tiếp và đồng bộ các hoạt động của chúng mà không chia sẻ cùng không gian địa chỉ. IPC đặc biệt có ích trong môi trường phân tán nơi các tiến trình giao tiếp có thể thường trú trên các máy tính khác được nối kết qua mạng. Thí dụ chương trình **chat** được dùng trên World Wide Web.

IPC được cung cấp bởi hệ thống truyền thông điệp, và các hệ thống truyền thông điệp có thể được định nghĩa trong nhiều cách. Trong phần này chúng ta sẽ xem xét những vấn đề khác nhau khi thiết kế các hệ thống truyền thông điệp.

V.1 Hệ thống truyền thông điệp

Chức năng của hệ thống truyền thông điệp là cho phép các tiến trình giao tiếp với các tiến trình khác mà không cần sắp xếp lại dữ liệu chia sẻ. Chúng ta xem truyền thông điệp được dùng như một phương pháp giao tiếp trong vi nhân. Trong cơ chế này, các dịch vụ được cung cấp như các tiến trình người dùng thông thường. Nghĩa là, các dịch vụ hoạt động bên ngoài nhân. Giao tiếp giữa các tiến trình người dùng được thực hiện thông qua truyền thông điệp. Một phương tiện IPC cung cấp ít nhất hai hoạt động: **send(message)** và **receive(message).**

Các thông điệp được gửi bởi một tiến trình có thể có kích thước cố định hoặc biến đổi. Nếu chỉ các thông điệp có kích thước cố định được gửi, việc cài đặt cấp hệ thống là đơn giản hơn. Tuy nhiên, hạn chế này làm cho tác vụ lập trình sẽ phức tạp hơn. Ngoài ra, các thông điệp có kích thước thay đổi yêu cầu việc cài đặt mức hệ thống phức tạp hơn nhưng tác vụ lập trình trở nên đơn giản hơn.

Nếu tiến trình P và Q muốn giao tiếp, chúng phải gửi các thông điệp tới và nhận thông điệp từ với nhau; một liên kết giao tiếp phải tồn tại giữa chúng. Liên kết này có thể được cài đặt trong những cách khác nhau. Ở đây chúng ta quan tâm đến cài đặt logic hơn là cài đặt vật lý. Có vài phương pháp cài đặt một liên kết và các hoạt động send/receive:

- Giao tiếp trực tiếp hay gián tiếp
- Giao tiếp đối xứng hay bất đối xứng
- Gửi bằng bản sao hay tham chiếu
- Thông điệp có kích thước cố định hay thay đổi

V.2 Đặt tên

Các tiến trình muốn giao tiếp phải có cách tham chiếu với nhau. Chúng có thể dùng giao tiếp trực tiếp hay gián tiếp.

V.2.1 Giao tiếp trực tiếp

Với giao tiếp trực tiếp, mỗi tiến trình muốn giao tiếp phải đặt tên rõ ràng người gửi và người nhận của giao tiếp. Trong cơ chế này, các hàm cơ sở send và receive được định nghĩa như sau:

- **Send(P, message):**gửi một thông điệp tới tiến trình P
- **Receive(Q, message):**nhận một thông điệp từ tiến trình Q

Một liên kết giao tiếp trong cơ chế này có những thuộc tính sau:

- Một liên kết được thiết lập tự động giữa mỗi cặp tiến trình muốn giao tiếp.
- Các tiến trình cần biết định danh của nhau khi giao tiếp.
- Một liên kết được nối kết với chính xác hai tiến trình
- Chính xác một liên kết tồn tại giữa mỗi cặp tiến trình.

Cơ chế này hiển thị tính đối xứng trong việc đánh địa chỉ: nghĩa là, cả hai quá trình gửi và nhận phải biết tên nhau để giao tiếp. Một thay đổi trong cơ chế này thực hiện tính bất đối xứng trong việc đánh địa chỉ. Chỉ người gửi biết tên của người nhận; người nhận không yêu cầu tên của người gửi. Trong cơ chế này các hàm cơ sở được định nghĩa như sau:

- **Send(P, message):** gửi một thông điệp tới tiến trình P
- **Receive(id, message):** nhận một thông điệp từ bất kỳ tiến trình nào; id khác nhau được đặt tên của tiến trình mà giao tiếp xảy ra.

Sự bất lợi trong cả hai cơ chế đối xứng và không đối xứng là tính điều chỉnh của việc định nghĩa tiến trình bị giới hạn. Thay đổi tên của một tiến trình có thể cần xem xét tất cả định nghĩa tiến trình khác. Tất cả tham chiếu tới tên cũ phải được tìm thấy để mà chúng có thể được thay đổi thành tên mới. Trường hợp này là không mong muốn từ quan điểm biên dịch riêng.

V.2.2 Giao tiếp gián tiếp

Với giao tiếp gián tiếp, một thông điệp được gửi tới và nhận từ các hộp thư (mailboxes), hay cổng (ports). Một hộp thư có thể được hiển thị trừu tượng như một đối tượng trong đó các thông điệp có thể được đặt bởi các tiến trình và sau đó các thông điệp này có thể được xóa đi. Mỗi hộp thư có một định danh duy nhất. Trong cơ chế này, một tiến trình có thể giao tiếp với một vài tiến trình khác bằng một số hộp thư khác nhau. Hai tiến trình có thể giao tiếp chỉ nếu chúng chia sẻ cùng một hộp thư. Hàm cơ sở **send** và **receive** được định nghĩa như sau:

- **Send(A, message):**gửi một thông điệp tới hộp thư A.
- **Receive(A, message):**nhận một thông điệp từ hộp thư A.

Trong cơ chế này, một liên kết giao tiếp có các thuộc tính sau:

- Một liên kết được thiết lập giữa một cặp tiến trình chỉ nếu cả hai thành viên của cặp có một hộp thư được chia sẻ.
- Một liên kết có thể được nối kết với nhiều hơn hai tiến trình.
- Số các liên kết khác nhau có thể tồn tại giữa mỗi cặp tiến trình giao tiếp với mỗi liên kết tương ứng với một hộp thư

Giả sử các tiến trình P_1, P_2 và P_3 chia sẻ một hộp thư A. Tiến trình P_1 gửi một thông điệp tới A trong khi P_2 và P_3 thực thi việc nhận từ A. Tiến trình nào sẽ nhận thông điệp được gửi bởi P_1? Câu trả lời phụ thuộc cơ chế mà chúng ta chọn:

- Cho phép một liên kết được nối kết với nhiều nhất hai tiến trình

• Cho phép nhiều nhất một tiến trình tại một thời điểm thực thi thao tác nhận.

• Cho phép hệ thống chọn bất kỳ tiến trình nào sẽ nhận thông điệp (nghĩa là, hoặc P_1 hoặc P_3 nhưng không phải cả hai sẽ nhận thông điệp). Hệ thống này có thể xác định người nhận tới người gửi.

Một hộp thư có thể được sở hữu bởi một tiến trình hay bởi hệ điều hành. Nếu hộp thư được sở hữu bởi một tiến trình (nghĩa là, hộp thư là một phần không gian địa chỉ của tiến trình), sau đó chúng ta phân biệt giữa người sở hữu (người chỉ nhận thông điệp thông qua hộp thư này) và người dùng (người có thể chỉ gửi thông điệp tới hộp thư).

Vì mỗi hộp thư có một người sở hữu duy nhất nên không có sự lẫn lộn về người nhận thông điệp được gửi tới hộp thư này. Khi một tiến trình sở hữu một hộp thư kết thúc, hộp thư biến mất. Sau đó, bất kỳ tiến trình nào gửi thông điệp tới hộp thư này được thông báo rằng hộp thư không còn tồn tại nữa.

Ngoài ra, một hộp thư được sở hữu bởi hệ điều hành độc lập và không được gán tới bất kỳ tiến trình xác định nào. Sau đó, hệ điều hành phải cung cấp một cơ chế cho phép một tiến trình thực hiện như sau:

• Tạo một hộp thư mới

• Gửi và nhận các thông điệp thông qua hộp thư

• Xóa hộp thư

Mặc định, tiến trình tạo hộp thư mới là người sở hữu hộp thư đó. Ban đầu, người sở hữu chỉ là một tiến trình có thể nhận thông điệp thông qua hộp thư. Tuy nhiên, việc sở hữu và quyền nhận thông điệp có thể được chuyển tới các tiến trình khác thông qua lời gọi hệ thống hợp lý. Dĩ nhiên, sự cung cấp này có thể dẫn đến nhiều người nhận cho mỗi hộp thư.

V.2.3 Đồng bộ hóa

Giao tiếp giữa hai tiến trình xảy ra bởi lời gọi hàm cơ sở **send** và **receive.** Có các tùy chọn thiết kế khác nhau cho việc cài đặt mỗi hàm cơ sở. Truyền thông điệp có thể là nghẽn (block) hay không nghẽn (nonblocking)-cũng được xem như đồng bộ và bất đồng bộ.

• **Hàm send nghẽn:** tiến trình gửi bị nghẽn cho đến khi thông điệp được nhận bởi tiến trình nhận hay bởi hộp thư.

• **Hàm send không nghẽn:** tiến trình gửi gửi thông điệp và thực hiện tiếp hoạt động

• **Hàm receive nghẽn:** người nhận nghẽn cho đến khi thông điệp sẵn dùng

• **Hàm receive không nghẽn:** người nhận nhận lại một thông điệp hợp lệ hay rỗng

Sự kết hợp khác nhau giữa send và receive là có thể. Khi cả hai send và receive là nghẽn chúng ta có sự thống nhất giữa người gửi và người nhận.

V.2.4 Tạo vùng đệm

Dù giao tiếp có thể là trực tiếp hay gián tiếp, các thông điệp được chuyển đổi bởi các tiến trình giao tiếp thường trú trong một hàng đợt tạm thời. Về cơ bản, một hàng đợi như thế có thể được cài đặt trong ba cách:

• **Khả năng chứa là 0** (zero capacity): hàng đợi có chiều dài tối đa là 0; do đó liên kết không thể có bất kỳ thông điệp nào chờ trong nó. Trong trường hợp này, người gửi phải nghẽn cho tới khi người nhận nhận thông điệp.

• **Khả năng chứa có giới hạn** (bounded capacity): hàng đợi có chiều dài giới hạn *n;* do đó, nhiều nhất n thông điệp có thể thường trú trong nó. Nếu hàng đợi không đầy khi một thông điệp mới được gửi, sau đó nó được đặt trong hàng đợi (thông điệp được chép hay một con trỏ thông điệp được giữ) và người gửi có thể tiếp tục thực thi không phải chờ. Tuy nhiên, liên kết có khả năng chứa giới hạn. Nếu một liên kết đầy, người gửi phải nghẽn cho tới khi không gian là sẵn dùng trong hàng đợi.

• **Khả năng chứa không giới hạn** (unbounded capacity): Hàng đợi có khả năng có chiều dài không giới hạn; do đó số lượng thông điệp bất kỳ có thể chờ trong nó. Người gửi không bao giờ nghẽn.

Trường hợp khả năng chứa là 0 thường được xem như hệ thống thông điệp không có vùng đệm; hai trường hợp còn lại được xem như vùng đệm tự động.

Ghi nhớ.

Tiến trình là một chương trình đang thực thi. Khi một tiến trình thực thi, nó thay đổi trạng thái. Trạng thái của một tiến trình được định nghĩa bởi một hoạt động hiện tại của tiến trình đó. Mỗi tiến trình có thể ở một trong những trạng thái sau: mới (new), sẵn sàng (ready), đang chạy (running), chờ (waiting), hay kết thúc (terminated). Mỗi tiến trình được biểu diễn trong hệ điều hành bởi khối điều khiển quá trình của chính nó (PCB).

Một tiến trình khi không thực thi, được đặt vào hàng đợi. Hai cấp chủ yếu của hàng đợi trong hệ điều hành là hàng đợi yêu cầu nhập/xuất và hàng đợi sẵn sàng. Hàng đợi sẵn sàng chứa tất cả tiến trình sẵn sàng để thực thi và đang chờ CPU. Mỗi tiến trình được biểu diễn bởi một PCB và các PCB có thể được liên kết với nhau để hình thành một hàng đợi sẵn sàng. **Định lịch biểu dài** (long-term scheduling) (hay định lịch biểu công việc) là chọn các tiến trình được phép cạnh tranh CPU. Thông thường, định lịch biểu dài bị ảnh hưởng nặng nề bởi việc xem xét cấp phát tài nguyên, đặc biệt quản lý bộ nhớ. **Lập lịch ngắn** (short-term scheduling) là sự chọn lựa một tiến trình từ các hàng đợi sẵn sàng.

Các tiến trình trong hệ thống có thể thực thi đồng hành. Có nhiều lý do các thực thi đồng hành: chia sẻ thông tin, tăng tốc độ tính toán, hiệu chỉnh và tiện dụng. Thực thi đồng hành yêu cầu cơ chế cho việc tạo và xóa tiến trình.

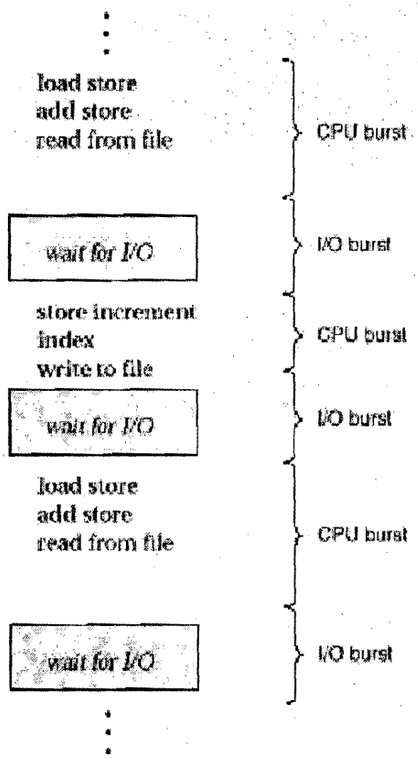
Tiến trình thực thi trong hệ điều hành có thể là các tiến trình độc lập hay các quá trình hợp tác. Các tiến trình hợp tác phải có phương tiện giao tiếp với nhau. Chủ yếu, có hai cơ chế giao tiếp bổ sung cho nhau cùng tồn tại: chia sẻ bộ nhớ và hệ thống truyền thông điệp. Phương pháp chia sẻ bộ nhớ yêu cầu các tiến trình giao tiếp chia sẻ một số biến. Các tiến trình được mong đợi trao đổi thông tin thông qua việc sử dụng các biến dùng chung này. Trong hệ thống bộ nhớ được chia sẻ, nhiệm vụ cho việc cung cấp giao tiếp tách rời với người lập trình ứng dụng; chỉ hệ điều hành cung cấp hệ thống bộ nhớ được chia sẻ. Phương pháp truyền thông điệp cho phép các tiến trình trong đối thông điệp. Nhiệm vụ cung cấp giao tiếp có thể tách rời với hệ điều hành.

Hai cơ chế này không loại trừ hỗ tương và có thể được dùng cùng một lúc trong phạm vi một hệ điều hành.

VI. Lập lịch thực thi tiến trình.

Lập lịch thực thi tiến trình là cơ sở của các hệ điều hành đa chương. Bằng cách chuyển đổi CPU giữa các tiến trình (từ bây giờ ta gọi là lập lịch CPU cho gọn), hệ điều hành có thể làm máy tính hoạt động nhiều hơn. Trong phần này, chúng ta giới thiệu các khái niệm lập lịch cơ bản và trình bày các giải thuật lập lịch thực thi tiến trình khác nhau. Chúng ta cũng xem xét vấn đề chọn một giải thuật cho một hệ thống xác định.

VI.1. Các khái niệm cơ bản



Mục tiêu của đa chương là có nhiều tiến trình chạy cùng thời điểm để tối ưu hóa việc sử dụng CPU. Trong hệ thống đơn xử lý, chỉ một tiến trình có thể chạy tại một thời điểm; bất cứ tiến trình nào khác đều phải chờ cho đến khi CPU rảnh và có thể được lập lịch lại. Ý tưởng của đa chương là tương đối đơn giản. Một tiến trình được thực thi cho đến khi nó phải chờ yêu cầu nhập/xuất hoàn thành. Trong một hệ thống máy tính đơn giản thì CPU sẽ rảnh rỗi; tất cả thời gian chờ này là lãng phí. Với đa chương, chúng ta cố gắng dùng thời gian này để CPU có thể phục vụ cho các tiến trình khác. Nhiều tiến trình được giữ trong bộ nhớ tại cùng thời điểm. Khi một tiến trình phải chờ, hệ điều hành lấy CPU từ tiến trình này và cấp CPU tới tiến trình khác. Lập lịch là chức năng cơ bản của hệ điều hành. Hầu hết tài nguyên máy tính được lập lịch trước khi dùng. Dĩ nhiên, CPU là một trong những tài nguyên máy tính ưu tiên. Do đó, lập lịch là trọng tâm trong việc thiết kế hệ điều hành.

Hình 3.7-Thay đổi thứ tự của CPU và I/O burst

VI.1.1. Chu kỳ CPU-I/O

Sự thành công của việc lập lịch CPU phụ thuộc vào thuộc tính được xem xét sau đây của tiến trình. Việc thực thi tiến trình chứa một **chu kỳ** (cycle) thực thi CPU và chờ đợi nhập/xuất. Các tiến trình chuyển đổi giữa hai trạng thái này. Sự thực thi tiến trình bắt đầu với một **chu kỳ CPU** (CPU burst), theo sau bởi một **chu kỳ nhập/xuất** (I/O burst), sau đó một chu kỳ CPU khác, sau đó lại tới một chu kỳ nhập/xuất khác khác,..Sau cùng, chu kỳ CPU cuối cùng sẽ kết thúc với một yêu cầu hệ thống để kết thúc việc thực thi, hơn là với một chu kỳ nhập/xuất khác, được mô tả như hình 3.7. Một chương trình **hướng nhập/xuất** (I/O-bound) thường có nhiều chu kỳ CPU ngắn. Một chương trình **hướng xử lý** (CPU-bound) có thể có một nhiều chu kỳ CPU dài. Sự phân bổ này có thể giúp chúng ta chọn giải thuật lập lịch CPU hợp lý.

VI.1.2 Bộ lập lịch CPU

Bất cứ khi nào CPU rảnh, hệ điều hành phải chọn một trong những tiến trình trong hàng đợi sẵn sàng để thực thi. Chọn tiến trình được thực hiện bởi **bộ lập lịch ngắn** (short-term scheduler) hay bộ định thời CPU. Bộ lập lịch này chọn các tiến trình trong bộ nhớ sẵn sàng thực thi và cấp phát CPU tới một trong các tiến trình đó.

Hàng đợi sẵn sàng không nhất thiết là hàng đợi vào trước, ra trước (FIFO). Xem

xét một số giải thuật lập lịch khác nhau, một hàng đợi sẵn sàng có thể được cài đặt như một hàng đợi FIFO, một hàng đợi ưu tiên, một cây, hay đơn giản là một danh sách liên kết không thứ tự. Tuy nhiên, về khái niệm tất cả các tiến trình trong hàng đợi sẵn sàng được xếp hàng chờ cơ hội để chạy trên CPU. Các mẫu tin trong hàng đợi thường là khối điều khiển tiến trình của tiến trình đó.

VI.1.3 Lập lịch trưng dụng

Quyết định lập lịch CPU có thể xảy ra một trong 4 trường hợp sau:

- Khi một tiến trình chuyển từ trạng thái chạy sang trạng thái chờ (thí dụ: yêu cầu nhập/xuất, hay chờ kết thúc của một trong những tiến trình con).
- Khi một tiến trình chuyển từ trạng thái chạy tới trạng thái sẵn sàng (thí dụ: khi một ngắt xảy ra)
- Khi một tiến trình chuyển từ trạng thái chờ tới trạng thái sẵn sàng (thí dụ: hoàn thành nhập/xuất)
- Khi một tiến trình kết thúc

Trong trường hợp 1 và 4, không cần chọn lựa loại lập lịch. Một tiến trình mới (nếu tồn tại trong hàng đợi sẵn sàng) phải được chọn để thực thi. Tuy nhiên, có sự lựa chọn loại lập lịch trong trường hợp 2 và 3.

Khi lập lịch xảy ra chỉ trong trường hợp 1 và 4, chúng ta nói cơ chế **lập lịch không trưng dụng** (nonpreemptive); ngược lại, khi lập lịch xảy ra chỉ trong trường hợp 2 và 3, chúng ta nói cơ chế **lập lịch trưng dụng** (preemptive).

Trong lập lịch không trưng dụng, một khi CPU được cấp phát tới một tiến trình, quá trình giữ CPU cho tới khi nó giải phóng CPU hay bởi kết thúc hay bởi chuyển tới trạng thái sẵn sàng. Phương pháp lập lịch này được dùng bởi các hệ điều hành Microsoft Windows 3.1 và bởi Apple Macintosh. Phương pháp này chỉ có thể được dùng trên các nền tảng phần cứng xác định vì nó không đòi hỏi phần cứng đặc biệt (thí dụ, một bộ đếm thời gian) được yêu cầu để lập lịch trưng dụng.

Tuy nhiên, lập lịch trưng dụng sinh ra một chi phí. Xét trường hợp 2 tiến trình chia sẻ dữ liệu. Một tiến trình có thể ở giữa giai đoạn cập nhật dữ liệu thì nó bị chiếm dụng CPU và một tiến trình thứ hai đang chạy. Tiến trình thứ hai có thể đọc dữ liệu mà nó hiện đang ở trong trạng thái thay đổi. Do đó, những kỹ thuật mới được yêu cầu để điều phối việc truy xuất tới dữ liệu được chia sẻ.

Sự trưng dụng cũng có một ảnh hưởng trong thiết kế nhân hệ điều hành. Trong khi xử lý lời gọi hệ thống, nhân có thể chờ một hoạt động dựa theo hành vi của quá trình. Những hoạt động như thế có thể liên quan với sự thay đổi dữ liệu nhân quan trọng (thí dụ: các hàng đợi nhập/xuất). Điều gì xảy ra nếu tiến trình bị trưng dụng CPU ở trong giai đoạn thay đổi này và nhân (hay trình điều khiển thiết bị) cần đọc hay sửa đổi cùng cấu trúc? Sự lộn xộn chắc chắn xảy ra. Một số hệ điều hành, gồm hầu hết các ấn bản của UNIX, giải quyết vấn đề này bằng cách chờ lời gọi hệ thống hoàn thành hay việc nhập/xuất bị nghẽn, trước khi chuyển đổi ngữ cảnh. Cơ chế này đảm bảo rằng cấu trúc nhân là đơn giản vì nhân sẽ không trưng dụng một tiến trình trong khi các cấu trúc dữ liệu nhân ở trong trạng thái thay đổi. Tuy nhiên, mô hình thực thi nhân này là mô hình nghèo nàn để hỗ trợ tính toán thời thực và đa xử lý.

Trong trường hợp UNIX, các phần mã vẫn là sự rủi ro. Vì các ngắt có thể xảy ra bất cứ lúc nào và vì các ngắt này không thể luôn được bỏ qua bởi nhân, nên phần mã bị ảnh hưởng bởi ngắt phải được đảm bảo từ việc sử dụng đồng thời. Hệ điều hành

cần chấp nhận hầu hết các ngắt, ngược lại dữ liệu nhập có thể bị mất hay dữ liệu xuất bị viết chồng. Vì thế các phần mã này không thể được truy xuất đồng hành bởi nhiều tiến trình, chúng vô hiệu hóa ngắt tại lúc nhập và cho phép các ngắt hoạt động trở lại tại thời điểm việc nhập kết thúc. Tuy nhiên, vô hiệu hóa và cho phép ngắt tiêu tốn thời gian, đặc biệt trên các hệ thống đa xử lý.

VI.1.4. Bộ phân phát

Một thành phần khác liên quan đến chức năng lập lịch CPU là **bộ phân phát** (dispatcher). Bộ phân phát là một module có nhiệm vụ trao điều khiển CPU tới tiến trình được chọn bởi bộ lập lịch ngắn (short-term scheduler). Chức năng này liên quan:

- Chuyển ngữ cảnh
- Chuyển chế độ người dùng
- Nhảy tới vị trí hợp lý trong chương trình người dùng để khởi động lại quá trình

Bộ phân phát nên nhanh nhất có thể, và nó được nạp trong mỗi lần chuyển tiến trình. Thời gian mất cho bộ phân phát dừng một tiến trình này và bắt đầu chạy một tiến trình khác được gọi là thời gian trễ cho việc điều phối (dispatch latency).

VI.2. Các tiêu chuẩn lập lịch

Các giải thuật lập lịch khác nhau có các thuộc tính khác nhau và có xu hướng thiên vị cho một loại tiến trình hơn một tiến trình. Trong việc chọn giải thuật nào sử dụng trong trường hợp nào, chúng ta phải xét các thuộc tính của các giải thuật khác nhau. Nhiều tiêu chuẩn được đề nghị để so sánh các giải thuật lập lịch. Những đặc điểm được dùng để so sánh có thể tạo sự khác biệt quan trọng trong việc xác định giải thuật tốt nhất. Các tiêu chuẩn gồm:

- **Việc sử dụng CPU:** chúng ta muốn giữ CPU bận nhiều nhất có thể. Việc sử dụng CPU có thể từ 0 đến 100%. Trong hệ thống thực, nó nên nằm trong khoảng từ 40% (cho hệ thống được nạp tải nhẹ) tới 90% (cho hệ thống được nạp tải nặng).
- **Thông lượng:** nếu CPU bận thực thi các tiến trình thì công việc đang được thực hiện. Thước đo của công việc là số lượng tiến trình được hoàn thành trên một đơn vị thời gian gọi là **thông lượng** (throughput). Đối với các quá trình dài, tỉ lệ này có thể là 1 tiến trình trên 1 giờ; đối với các giao dịch ngắn, thông lượng có thể là 10 tiến trình trên giây.
- **Thời gian hoàn thành:** từ quan điểm của một tiến trình cụ thể, tiêu chuẩn quan trọng là mất bao lâu để thực thi tiến trình đó. Khoảng thời gian từ thời điểm gửi tiến trình tới khi tiến trình hoàn thành được gọi là **thời gian hoàn thành** (turnaround time). Thời gian hoàn thành là tổng các thời gian chờ đưa tiến trình vào bộ nhớ, chờ hàng đợi sẵn sàng, thực thi CPU và thực hiện nhập/xuất.
- **Thời gian chờ:** giải thuật lập lịch CPU không ảnh hưởng lượng thời gian tiến trình thực thi hay thực hiện nhập/xuất; nó ảnh hưởng chỉ lượng thời gian một tiến trình phải chờ trong hàng đợi sẵn sàng. **Thời gian chờ** (waiting time) là tổng thời gian chờ trong hàng đợi sẵn sàng.
- **Thời gian đáp ứng:** trong một hệ thống giao tiếp, thời gian hoàn thành không là tiêu chuẩn tốt nhất. Thông thường, một tiến trình có thể tạo ra

dữ liệu xuất tương đối sớm và có thể tiếp tục tính toán các kết quả mới trong khi các kết quả trước đó đang được xuất cho người dùng. Do đó, một thước đo khác là thời gian từ lúc gởi yêu cầu cho tới khi đáp ứng đầu tiên được tạo ra. Thước đo này được gọi là **thời gian đáp ứng** (response time), là lượng thời gian mất đi từ lúc bắt đầu đáp ứng nhưng không là thời gian mất đi để xuất ra đáp ứng đó. Thời gian hoàn thành thường bị giới hạn bởi tốc độ của thiết bị xuất.

Chúng ta muốn tối ưu hóa việc sử dụng CPU và thông lượng, đồng thời tối thiểu hóa thời gian hoàn thành, thời gian chờ, và thời gian đáp ứng. Trong hầu hết các trường hợp, chúng ta tối ưu hóa thước đo trung bình. Tuy nhiên, trong một vài trường hợp chúng ta muốn tối ưu giá trị tối thiểu hay giá trị tối đa hơn là giá trị trung bình. Thí dụ, để đảm bảo rằng tất cả người dùng nhận dịch vụ tốt, chúng ta muốn tối thiểu thời gian đáp ứng tối đa.

Đối với những hệ thống tương tác (như các hệ thống chia thời gian), một số nhà phân tích đề nghị rằng sự thay đổi trong thời gian đáp ứng quan trọng hơn tối thiểu hóa thời gian đáp ứng trung bình. Một hệ thống với thời gian đáp ứng phù hợp và có thể đoán trước được quan tâm nhiều hơn hệ thống chạy nhanh hơn mức trung bình nhưng biến đổi cao. Tuy nhiên, gần như không có công việc nào được thực hiện trên các giải thuật lập lịch CPU để tối thiểu hóa các thay đổi.

Khi chúng ta thảo luận các giải thuật lập lịch CPU khác nhau, chúng ta muốn hiển thị các hoạt động của chúng. Một hình ảnh chính xác nên thông báo tới nhiều tiến trình, mỗi tiến trình là một chuỗi của hàng trăm chu kỳ CPU và I/O. Để đơn giản việc hiển thị, chúng ta xem chỉ một chu kỳ CPU (trong mili giây) trên tiến trình trong các thí dụ của chúng ta. Thước đo của việc so sánh là thời gian chờ đợi trung bình.

VI.3. Các giải thuật lập lịch

Lập lịch CPU giải quyết vấn đề quyết định tiến trình nào trong hàng đợi sẵn sàng được cấp phát CPU. Trong phần này chúng ta mô tả nhiều giải thuật định thời CPU đang có.

VI.3.1. Lập lịch đến trước được phục vụ trước

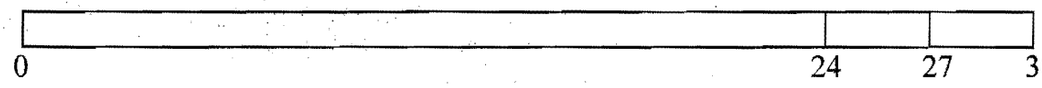
Giải thuật lập lịch CPU đơn giản nhất là **đến trước, được phục vụ trước** (first-come, first-served: FCFS). Với cơ chế này, tiến trình yêu cầu CPU trước được cấp phát CPU trước. Việc cài đặt chính sách FCFS được quản lý dễ dàng với hàng đợi FIFO. Khi một tiến trình đi vào hàng đợi sẵn sàng, PCB của nó được liên kết tới đuôi của hàng đợi. Khi CPU rảnh, nó được cấp phát tới một tiến trình tại đầu hàng đợi. Sau đó, tiến trình đang chạy được lấy ra khỏi hàng đợi. Mã của giải thuật FCFS đơn giản để viết và hiểu. Tuy nhiên, thời gian chờ đợi trung bình dưới chính sách FCFS thường là dài.

Xét tập hợp các tiến trình sau đến tại thời điểm 0, với chiều dài thời gian chu kỳ CPU được cho theo mili giây (hoặc một đơn vị thời gian nào đó).

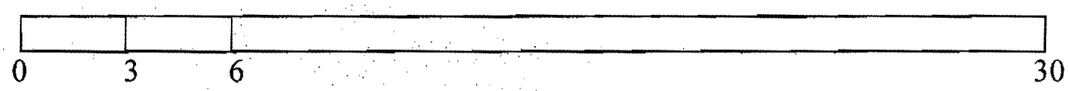
Tiến trình	Thời gian xử lý
P1	24
P2	3
P3	3

Nếu các tiến trình đến theo thứ tự P_1, P_2, P_3 và được phục vụ theo thứ tự FCFS,

chúng ta nhận được kết quả được hiển thị trong **lưu đồ Gantt** như sau:



Thời gian chờ là 0 mili giây cho tiến trình P_1, 24 mili giây cho tiến trình P_2 và 27 mili giây cho tiến trình P_3. Do đó, thời gian chờ đợi trung bình là (0+24+27)/3=17mili giây. Tuy nhiên, nếu các tiến trình đến theo thứ tự P_2, P_3, P_1 thì các kết quả được hiển thị trong **lưu đồ Gannt** như sau:



Thời gian chờ đợi trung bình bây giờ là (6+0+3)/3=3 mili giây. Việc cắt giảm này là quan trọng. Do đó, thời gian chờ đợi trung bình dưới chính sách FCFS thường không là tối thiểu và có sự thay đổi rất quan trọng nếu các thời gian CPU dành cho các tiến trình khác nhau rất lớn.

Ngoài ra, xét năng lực của lập lịch FCFS trong trường hợp động. Giả sử chúng ta có một tiến trình hướng xử lý (CPU-bound) và nhiều tiến trình hướng nhập/xuất (I/O bound). Khi các tiến trình đưa đến quanh hệ thống, ngữ cảnh sau có thể xảy ra. Tiến trình hướng xử lý sẽ nhận CPU và giữ nó. Trong suốt thời gian này, tất cả tiến trình khác sẽ kết thúc việc nhập/xuất của nó và chuyển vào hàng đợi sẵn sàng, các thiết bị nhập/xuất ở trạng thái rảnh. Cuối cùng, tiến trình hướng xử lý kết thúc chu kỳ CPU của nó và chuyển tới thiết bị nhập/xuất. Tất cả các tiến trình hướng xử lý có chu kỳ CPU rất ngắn sẽ nhanh chóng thực thi và di chuyển trở về hàng đợi nhập/xuất. Tại thời điểm này CPU ở trạng thái rảnh. Sau đó, tiến trình hướng xử lý sẽ di chuyển trở lại hàng đợi sẵn sàng và được cấp CPU. Một lần nữa, tất cả tiến trình hướng nhập/xuất kết thúc việc chờ trong hàng đợi sẵn sàng cho đến khi tiến trình hướng xử lý được thực hiện. Có một **tác dụng phụ** (convoy effect) khi tất cả các quá trình khác chờ một tiến trình lớn trả lại CPU. Tác dụng phụ này dẫn đến việc sử dụng thiết bị và CPU thấp hơn nếu các tiến trình ngắn hơn được cấp trước.

Giải thuật FCSF là giải thuật lập lịch không trưng dụng CPU. Một khi CPU được cấp phát tới một tiến trình, tiến trình đó giữ CPU cho tới khi nó giải phóng CPU bằng cách kết thúc hay yêu cầu nhập/xuất. Giải thuật FCFS đặc biệt không phù hợp đối với hệ thống chia sẻ thời gian, ở đó mỗi người dùng nhận được sự chia sẻ CPU với những khoảng thời gian đều nhau.

VI.3.2. Lập lịch công việc ngắn nhất trước

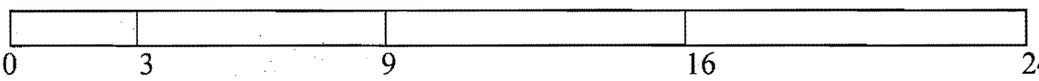
Một tiếp cận khác đối với việc lập lịch CPU là giải thuật lập lịch **công việc ngắn nhất trước** (shortest-job-first: SJF). Giải thuật này gán tới mỗi tiến trình chiều dài của chu kỳ CPU tiếp theo cho tiến trình sau đó. Khi CPU sẵn dùng, nó được gán tới tiến trình có chu kỳ CPU kế tiếp ngắn nhất. Nếu hai tiến trình có cùng chiều dài chu kỳ CPU kế tiếp, lập lịch FCFS được dùng. Chú ý rằng thuật ngữ phù hợp hơn là chu kỳ CPU kế tiếp ngắn nhất (shortest next CPU burst) vì lập lịch được thực hiện bằng cách xem xét chiều dài của chu kỳ CPU kế tiếp của tiến trình hơn là toàn bộ chiều dài của nó. Chúng ta dùng thuật ngữ SJF vì hầu hết mọi người và mọi sách tham khảo tới nguyên lý của loại lập lịch này như SJF.

Thí dụ, xét tập hợp các tiến trình sau, với chiều dài của thời gian chu kỳ CPU được tính bằng mili giây:

Tiến trình	Thời gian xử lý

P1	6
P2	8
P3	7
P4	3

Dùng lập lịch SJF, chúng ta lập lịch cho các tiến trình này theo **lưu đồ Gannt** như sau:



Thời gian chờ đợi là 3 mili giây cho tiến trình P_1, 16 mili giây cho tiến trình P_2, 9 mili giây cho tiến trình P_3, và 0 mili giây cho tiến trình P_4. Do đó, thời gian chờ đợi trung bình là (3+16+9+0)/4 = 7 mili giây. Nếu chúng ta dùng cơ chế lập lịch FCFS thì thời gian chờ đợi trung bình là 10.23 mili giây.

Giải thuật SJF có thể là tối ưu, trong đó nó cho thời gian chờ đợi trung bình nhỏ nhất cho các tiến trình được cho. Bằng cách chuyển một tiến trình ngắn trước một tiến trình dài thì thời gian chờ đợi của tiến trình ngắn giảm hơn so với việc tăng thờigian chờ đợi của tiến trình dài. Do đó, thời gian chờ đợi trung bình giảm.

Khó khăn thật sự với giải thuật SJF là làm thế nào để biết chiều dài của yêu cầu CPU tiếp theo. Đối với lập lịch dài trong hệ thống lô, chúng ta có thể dùng chiều dài như giới hạn thời gian xử lý mà người dùng xác định khi gởi công việc. Do đó, người dùng được cơ động để ước lượng chính xác giới hạn thời gian xử lý vì giá trị thấp hơn có nghĩa là đáp ứng nhanh hơn. Lập lịch SJF được dùng thường xuyên trong lập lịch dài.

Mặc dù SJF là tối ưu nhưng nó không thể được cài đặt tại cấp lập lịch CPU ngắn vì không có cách nào để biết chiều dài chu kỳ CPU tiếp theo. Một tiếp cận là khác gần đúng lập lịch SJF được thực hiện. Chúng ta có thể không biết chiều dài của chu kỳ CPU kế tiếp nhưng chúng ta có đoán giá trị của nó. Chúng ta mong muốn rằng chu kỳ CPU kế tiếp sẽ tương tự chiều dài những chu kỳ CPU trước đó. Do đó, bằng cách tính toán mức xấp xỉ chiều dài của chu kỳ CPU kế tiếp, chúng ta chọn một quá trình với chu kỳ CPU được đoán là ngắn nhất.

Chu kỳ CPU kế tiếp thường được đoán như trung bình số mũ của chiều dài các chu kỳ CPU trước đó. Gọi t_n là chiều dài của chu kỳ CPU thứ n và gọi T_{n+1} giá trị được đoán cho chu kỳ CPU kế tiếp. Thì đối với α, với $0 \leq \alpha \leq 1$, định nghĩa

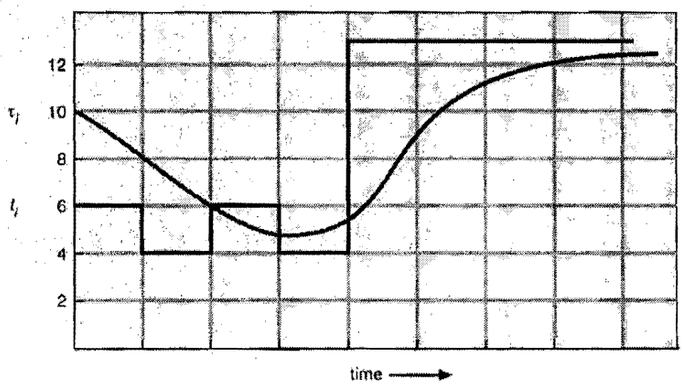
$$T_{n+1} = \alpha\, t_n + (1- \alpha)\, T_n$$

Công thức này định nghĩa một giá trị trung bình số mũ. Giá trị của t_n chứa thông tin mới nhất; T_n lưu lịch sử quá khứ. Tham số α điều khiển trọng số liên quan giữa lịch sử quá khứ và lịch sử gần đây trong việc đoán. Nếu α=0 thì $T_{n+1}=T_n$ và lịch sử gần đây không có ảnh hưởng (điều kiện hiện hành được đảm bảo là ngắn); nếu α =1 thì $T_{n+1}=t_n$ và chỉ chu kỳ CPU gần nhất có ảnh hưởng (lịch sử được đảm bảo là cũ và không phù hợp). Thông dụng hơn, α=1/2 thì lịch sử gần đây và lịch sử quá khứ có trọng số tương đương. Giá trị khởi đầu T_0 có thể được định nghĩa như một hằng số hay như toàn bộ giá trị trung bình hệ thống. Hình 3.8 dưới đây hiển thị giá trị trung bình dạng mũ với α=1/2 và T_0=10.

Để hiểu hành vi của giá trị trung bình dạng mũ, chúng ta có thể mở rộng công thức cho T_{n+1} bằng cách thay thế T_n để tìm

$$T_{n+1}=\alpha\, t_n+(1-\alpha)\,\alpha\, t_{n-1}+\ldots+(1-\alpha)^j\,\alpha\, t_{n-j}+\ldots+(1-\alpha)^{n-1}T_0$$

Vì cả hai α và (1- α) là nhỏ hơn hay bằng 1, mỗi số hạng tiếp theo có trọng số nhỏ hơn số hạng trước đó.



CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

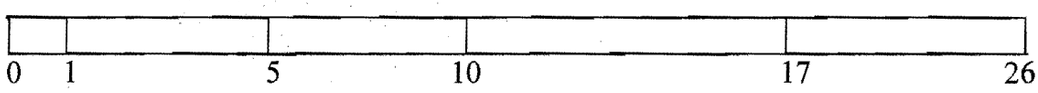
Hình 3.8. Đoán chiều dài của chu kỳ CPU kế tiếp

Giải thuật SJF có thể trưng dụng hoặc không trưng dụng CPU. Chọn lựa này phát sinh khi một tiến trình mới đến tại hàng đợi sẵn sàng trong khi một tiến trình trước đó đang thực thi. Một tiến trình mới có thể có chu kỳ CPU tiếp theo ngắn hơn chu kỳ CPU được để lại của tiến trình thực thi hiện tại. Giải thuật SJF trưng dụng sẽ trưng dụng CPU của tiến trình đang thực thi hiện tại, trong khi giải thuật SJF không trưng dụng sẽ cho phép tiến trình đang thực thi kết thúc chu kỳ CPU của nó. Lập lịch SJF trưng dụng còn được gọi là **lập lịch thời gian còn lại ngắn nhất trước** (shortest-remaining-time-first).

Thí dụ, xét 4 tiến trình sau với chiều dài của thời gian chu kỳ CPU được cho tính bằng mili giây.

Tiến trình	Thời gian đến	Thời gian xử lý
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Nếu các tiến trình đi vào hàng đợi sẵn sàng tại những thời điểm và cần thời gian xử lý được hiển thị trong bảng trên thì thời biểu SJF trưng dụng được mô tả trong **lưu đồ Gannt** như sau:



Tiến trình P_1 được bắt đầu tại thời điểm 0, vì nó là tiến trình duy nhất trong hàng đợi. Tiến trình P_2 đến tại thời điểm 1. Thời gian còn lại cho P_1 (7 mili giây) là lớn hơn thời gian được yêu cầu bởi tiến trình P_2 (4 mili giây) vì thế tiến trình P_1 bị trưng dụng CPU và tiến trình P_2 được lập lịch. Thời gian chờ đợi trung bình cho thí dụ này là: ((10-1) + (1-1) + (17-2) + (5-3))/4 = 6.5 mili giây. Lập lịch SJF không trưng dụng cho kết quả thời gian chờ đợi trung bình là 7.75 mili giây.

VI.3.3. Lập lịch theo độ ưu tiên

Giải thuật SJF là trường hợp đặc biệt của **giải thuật lập lịch theo độ ưu tiên** (priority-scheduling algorithm). Độ ưu tiên được gán với mỗi tiến trình và CPU được cấp phát tới tiến trình với độ ưu tiên cao nhất. Tiến trình có độ ưu tiên bằng nhau được lập lịch trong thứ tự FCFS.

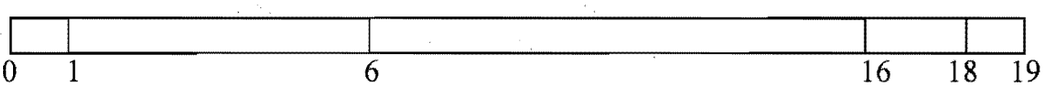
Giải thuật SJF là giải thuật ưu tiên đơn giản ở đó độ ưu tiên **p** là nghịch đảo với chu kỳ CPU được đoán tiếp theo. Chu kỳ CPU lớn hơn có độ ưu tiên thấp hơn và ngược lại.

Bây giờ chúng ta thảo luận lập lịch có độ ưu tiên cao và thấp. Các độ ưu tiên thường nằm trong dãy số cố định, chẳng hạn 0 tới 7 hay 0 tới 4,095. Tuy nhiên, không có sự thoả thuận chung về 0 là độ ưu tiên thấp nhất hay cao nhất. Một vài hệ thống dùng số thấp để biểu diễn độ ưu tiên thấp; ngược lại các hệ thống khác dùng các số thấp cho độ ưu tiên cao. Sự khác nhau này có thể dẫn đến sự lẫn lộn. Trong giáo trình này chúng ta dùng các số thấp để biểu diễn độ ưu tiên cao.

Thí dụ, xét tập hợp tiến trình sau đến tại thời điểm 0 theo thứ tự P_1, P_2,..., P_5 với chiều dài thời gian chu kỳ CPU được tính bằng mili giây:

Tiến trình	Thời gian xử lý	Độ ưu tiên
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Sử dụng lập lịch theo độ ưu tiên, chúng ta sẽ lập lịch các tiến trình này theo **lưu đồ Gannt** như sau:



Thời gian chờ đợi trung bình là 8.2 mili giây. Độ ưu tiên có thể được định nghĩa bên trong hay bên ngoài. Độ ưu tiên được định nghĩa bên trong thường dùng định lượng hoặc nhiều định lượng có thể đo để tính toán độ ưu tiên của một tiến trình. Thí dụ, các giới hạn thời gian, các yêu cầu bộ nhớ, số lượng tập tin đang mở và tỉ lệ của chu kỳ nhập/xuất trung bình với tỉ lệ của chu kỳ CPU trung bình. Các độ ưu tiên bên ngoài được thiết lập bởi các tiêu chuẩn bên ngoài đối với hệ điều hành như sự quan trọng của tiến trình, loại và lượng chi phí đang được trả cho việc dùng máy tính, văn phòng hỗ trợ công việc, ..

Lập lịch theo độ ưu tiên có thể trưng dụng hoặc không trưng dụng CPU. Khi một tiến trình đến hàng đợi sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của tiến trình hiện đang chạy. Giải thuật lập lịch theo độ ưu tiên trưng dụng sẽ chiếm CPU nếu độ ưu tiên của tiến trình mới đến cao hơn độ ưu tiên của tiến trình đang thực thi. Giải thuật lập lịch theo độ ưu tiên không trưng dụng sẽ đơn giản đặt tiến trình mới tại đầu hàng đợi sẵn sàng.

Vấn đề chính với giải thuật lập lịch theo độ ưu tiên là **nghẽn không hạn định** (indefinite blocking) hay **chết đói CPU** (starvation). Một tiến trình sẵn sàng chạy nhưng thiếu CPU có thể xem như bị nghẽn-chờ đợi CPU. Giải thuật lập lịch theo độ ưu tiên có thể để lại nhiều tiến trình có độ ưu tiên thấp chờ CPU không hạn định.

Trong một hệ thống máy tính tải cao, dòng đều đặn các tiến trình có độ ưu tiên cao hơn có thể ngăn chặn việc nhận CPU của tiến trình có độ ưu tiên thấp.. Thông thường, một trong hai trường hợp xảy ra. Cuối cùng, một tiến trình sẽ được chạy (lúc 2 a.m chủ nhật là thời điểm cuối cùng hệ thống nạp các tiến trình nhẹ), hay cuối cùng hệ thống máy tính sẽ đổ vỡ và mất tất cả các tiến trình có độ ưu tiên thấp chưa được kết thúc.

Một giải pháp cho vấn đề nghẽn không hạn định này là **sự lão hóa** (aging). Láo hóa là kỹ thuật tăng dần độ ưu tiên của tiến trình chờ trong hệ thống một thời gian dài. Thí dụ, nếu các độ ưu tiên nằm trong dãy từ 127 (thấp) đến 0 (cao), chúng ta giảm độ ưu tiên của tiến trình đang chờ xuống 1 mỗi 15 phút. Cuối cùng, thậm chí một quá trình với độ ưu tiên khởi đầu 127 sẽ đạt độ ưu tiên cao nhất trong hệ thống và sẽ được thực thi. Thật vậy, một tiến trình sẽ mất không quá 32 giờ để đạt được độ ưu tiên từ 127 tới 0.

VI.3.4.Lập lịch luân phiên

Giải thuật lập lịch luân phiên (round-robin scheduling algorithm-RR) được thiết kế đặc biệt cho hệ thống chia sẻ thời gian. Tương tự như lập lịch FCFS nhưng sự trưng dụng CPU được thêm vào để chuyển CPU giữa các tiến trình. Đơn vị thời gian nhỏ được gọi là định mức thời gian (time quantum) hay phần thời gian (time slice) được định nghĩa. Định mức thời gian thường từ 10 đến 100 mili giây. Hàng đợi sẵn sàng được xem như một hàng đợi vòng. Bộ lập lịch CPU di chuyển vòng quanh hàng đợi sẵn sàng, cấp phát CPU tới mỗi tiến trình có khoảng thời gian tối đa bằng một định mức thời gian.

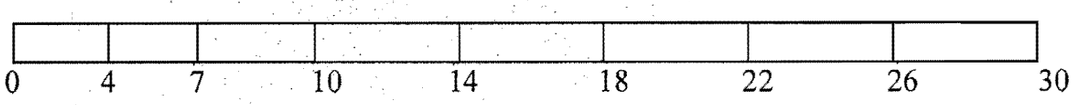
Để cài đặt lập lịch RR, chúng ta quản lý hàng đợi sẵn sàng như một hàng đợi FIFO của các tiến trình. Các tiến trình mới được thêm vào đuôi hàng đợi. Bộ lập lịch CPU chọn tiến trình đầu tiên từ hàng đợi sẵn sàng, đặt bộ đếm thời gian để ngắt sau 1 định mức thời gian và gởi tới tiến trình. Sau đó, một trong hai trường hợp sẽ xảy ra. Tiến trình có 1 chu kỳ CPU ít hơn 1 định mức thời gian. Trong trường hợp này, tiến trình sẽ tự giải phóng. Sau đó, bộ lập lịch sẽ xử lý tiến trình tiếp theo trong hàng đợi sẵn sàng. Ngược lại, nếu chu kỳ CPU của tiến trình đang chạy dài hơn 1 định mức thời gian thì độ đếm thời gian sẽ báo và gây ra một ngắt tới hệ điều hành. Chuyển đổi ngữ cảnh sẽ được thực thi và quá trình được đặt trở lại tại đuôi của hàng đợi sẵn sàng. Sau đó, bộ lập lịch CPU sẽ chọn tiến trình tiếp theo trong hàng đợi sẵn sàng.

Tuy nhiên, thời gian chờ đợi trung bình dưới chính sách RR thường là quá dài. Xét một tập hợp các tiến trình đến tại thời điểm 0 với chiều dài thời gian CPU-burst được tính bằng mili giây:

Tiến trình	Thời gian xử lý
P1	24
P2	3
P3	3

Nếu sử dụng định mức thời gian là 4 mili giây thì tiến trình P_1 nhận 4 mili giây đầu tiên. Vì nó yêu cầu 20 mili giây còn lại nên nó bị trưng dụng CPU sau định mức thời gian đầu tiên và CPU được cấp tới tiến trình tiếp theo trong hàng đợi, tiến trình P_2. Vì P_2 không cần tới 4 mili giây nên nó kết thúc trước khi định mức thời gian của nó hết hạn. Sau đó, CPU được cho tới tiến trình kế tiếp, tiến trình P_3. Một khi mỗi quá trình nhận 1 định mức thời gian thì CPU trả về tiến trình P_1 cho định mức thời gian tiếp theo.

Sơ đồ của RR là:

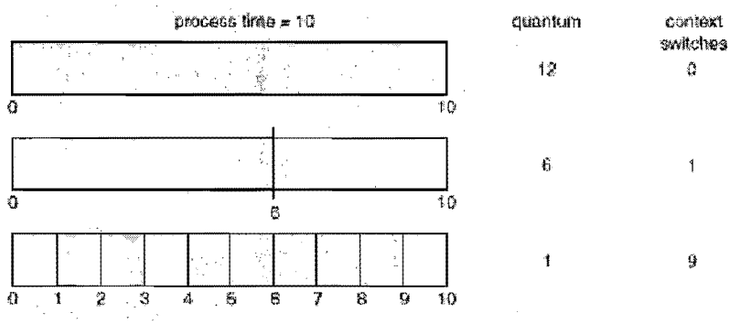


Thời gian chờ đợi trung bình là 17/3=5.66 mili giây. Trong giải thuật RR, không tiến trình nào được cấp phát CPU cho nhiều hơn 1 định mức thời gian trong một hàng. Nếu chu kỳ CPU của tiến trình vượt quá 1 định mức thời gian thì tiến trình đó bị trưng dụng CPU và nó được đặt trở lại hàng đợi sẵn sàng. Giải thuật RR là giải thuật trưng dụng CPU.

Nếu có n tiến trình trong hàng đợi sẵn sàng và định mức thời gian là q thì mỗi tiến trình nhận 1/n thời gian CPU trong các phần, nhiều nhất q đơn vị thời gian. Mỗi tiến trình sẽ chờ không dài hơn (n-1)x q đơn vị thời gian cho tới khi định mức thời gian tiếp theo của nó. Thí dụ, nếu có 5 tiến trình với định mức thời gian 20 mili giây thì mỗi tiến trình sẽ nhận 20 mili giây sau mỗi 100 mili giây.

Năng lực của giải thuật RR phụ thuộc nhiều vào kích thước của định mức thời gian. Nếu định mức thời gian rất lớn (lượng vô hạn) thì chính sách RR tương tự như chính sách FCFS. Nếu định mức thời gian là rất nhỏ (1 mili giây) thì tiếp cận RR được gọi là **chia sẻ bộ xử lý** (processor sharing) và xuất hiện (trong lý thuyết) tới người dùng như thể mỗi tiến trình trong n tiến trình có bộ xử lý riêng của chính nó chạy tại 1/n tốc độ của bộ xử lý thật.

Tuy nhiên, trong phần mềm chúng ta cũng cần xem xét hiệu quả của việc chuyển đổi ngữ cảnh trên năng lực của việc lập lịch RR. Chúng ta giả sử rằng chỉ có 1 tiến trình với 10 đơn vị thời gian. Nếu một định mức là 12 đơn vị thời gian thì quá trình kết thúc ít hơn 1 định mức thời gian, với không có chi phí nào khác. Tuy nhiên, nếu định mức là 6 đơn vị thời gian thì tiến trình cần 2 định mức thời gian, dẫn đến 1 chuyển đổi ngữ cảnh. Nếu định mức thời gian là 1 đơn vị thời gian thì 9 chuyển đổi ngữ cảnh sẽ xảy ra, việc thực thi của tiến trình bị chậm như được hiển thị trong hình 3.9

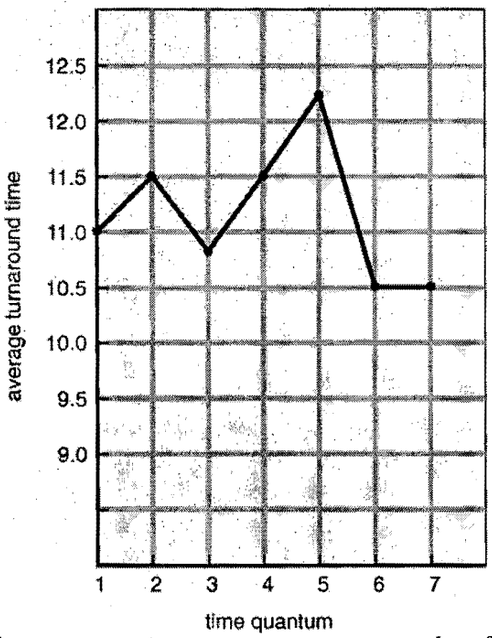


Hình 3.9. Quan hệ giữa định mức thời gian và thời gian chuyển đổi ngữ cảnh.

Do đó chúng ta mong muốn định mức thời gian lớn đối với thời gian chuyển ngữ cảnh. Nếu thời gian chuyển ngữ cảnh chiếm 10% định mức thời gian thì khoảng 10% thời gian CPU sẽ được dùng cho việc chuyển ngữ cảnh.

Thời gian hoàn thành cũng phụ thuộc kích thước của định mức thời gian. Chúng ta có thể thấy trong hình 3.10, thời gian hoàn thành trung bình của tập hợp các

tiến trình không cần cải tiến khi kích thước định mức thời gian tăng. Thông thường, thời gian hoàn thành trung bình có thể được cải tiến nếu hầu hết tiến trình kết thúc chu kỳ CPU kế tiếp của chúng trong một định mức thời gian. Thí dụ, cho 3 tiến trình có 10 đơn vị thời gian cho mỗi tiến trình và định mức thời gian là 1 đơn vị thời gian, thì thời gian hoàn thành trung bình là 29. Tuy nhiên, nếu định mức thời gian là 10 thì thời gian hoàn thành trung bình giảm tới 20. Nếu thời gian chuyển ngữ cảnh được thêm vào thì thời gian hoàn thành trung bình gia tăng đối với định mức thời gian nhỏ hơn vì các chuyển đổi ngữ cảnh thêm nữa sẽ được yêu cầu.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Hình 3.10. Hiển thị cách thời gian hoàn thành biến đổi theo định mức thời gian

Ngoài ra, nếu định mức thời gian quá lớn thì lập lịch RR trở thành lập lịch FCFS. Qui tắc là định mức thời gian nên dài hơn 80% chu kỳ CPU.

VI.3.5. Lập lịch với hàng đợi nhiều cấp

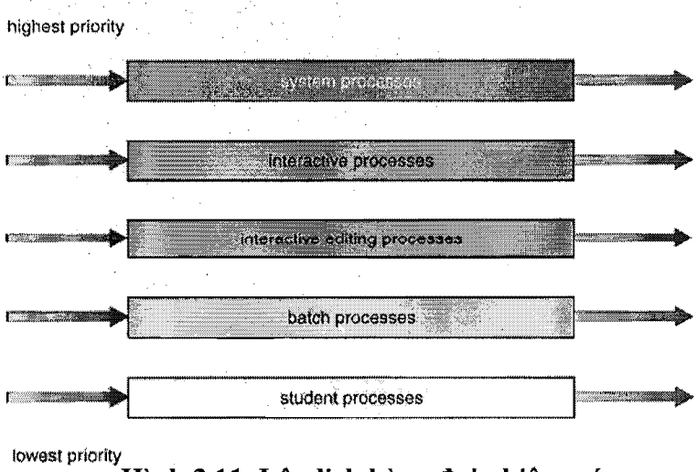
Một loại giải thuật lập lịch khác được tạo ra cho những trường hợp mà trong đó các tiến trình được phân lớp thành các nhóm khác nhau. Thí dụ: việc phân chia thông thường được thực hiện giữa các tiến trình chạy ở chế độ giao tiếp (foreground hay interactive) và các tiến trình chạy ở chế độ nền hay dạng lô (background hay batch). Hai loại tiến trình này có yêu cầu đáp ứng thời gian khác nhau và vì thế có yêu cầu về lập lịch khác nhau. Ngoài ra, các tiến trình chạy ở chế độ giao tiếp có độ ưu tiên (hay được định nghĩa bên ngoài) cao hơn các tiến trình chạy ở chế độ nền.

Một **giải thuật lập lịch hàng đợi nhiều cấp** (multilevel queue-scheduling algorithm) chia hàng đợi thành nhiều hàng đợi riêng rẻ (hình 3.11). Các tiến trình được gán vĩnh viễn tới một hàng đợi, thường dựa trên thuộc tính của tiến trình như kích thước bộ nhớ, độ ưu tiên tiến trình hay loại tiến trình. Mỗi hàng đợi có giải thuật lập lịch của chính nó. Thí dụ: các hàng đợi riêng rẻ có thể được dùng cho các quá trình ở chế độ nền và chế độ giao tiếp. Hàng đợi ở chế độ giao tiếp có thể được định thời bởi giải thuật RR trong khi hàng đợi ở chế độ nền được lập lịch bởi giải thuật FCFS.

Ngoài ra, phải có việc lập lịch giữa các hàng đợi, mà thường được cài đặt như lập lịch trưng dụng với độ ưu tiên cố định. Thí dụ, hàng đợi ở chế độ giao tiếp có độ ưu tiên tuyệt đối hơn hàng đợi ở chế độ nền.

Chúng ta xét một thí dụ của giải thuật hàng đợi nhiều mức với 5 hàng đợi:

- Các tiến trình hệ thống
- Các tiến trình giao tiếp
- Các tiến trình soạn thảo giao tiếp
- Các tiến trình lô
- Các tiến trình sinh viên



Hình 3.11. Lập lịch hàng đợi nhiều mức

Mỗi hàng đợi có độ ưu tiên tuyệt đối hơn hàng đợi có độ ưu tiên thấp hơn. Thí dụ: không có tiến trình nào trong hàng đợi lô có thể chạy trừ khi hàng đợi cho các quá trình hệ thống, các tiến trình giao tiếp và các tiến trình soạn thảo giao tiếp đều rỗng.

Nếu một tiến trình soạn thảo giao tiếp được đưa vào hàng đợi sẵn sàng trong khi một tiến trình lô đang chạy thì tiến trình lô bị trưng dụng CPU. Solaris 2 dùng dạng giải thuật này.

Một khả năng khác là phần (slice) thời gian giữa hai hàng đợi. Mỗi hàng đợi nhận một phần thời gian CPU xác định, sau đó nó có thể lập lịch giữa các tiến trình khác nhau trong hàng đợi của nó. Thí dụ, trong hàng đợi giao tiếp-nền, hàng đợi giao tiếp được cho 80% thời gian của CPU cho giải thuật RR giữa các tiến trình của nó, ngược lại hàng đợi nền nhận 20% thời gian CPU cho các tiến trình của nó theo cách FCFS.

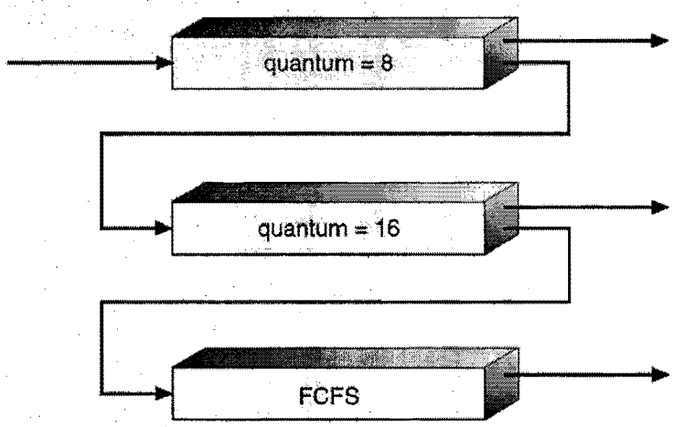
VI.3.6. Lập lịch hàng đợi phản hồi đa cấp

Thông thường, trong giải thuật hàng đợi đa cấp, các tiến trình được gán vĩnh viễn tới hàng đợi khi được đưa vào hệ thống. Các tiến trình không di chuyển giữa các hàng đợi. Nếu có các hàng đợi riêng cho các tiến trình giao tiếp và các tiến trình nền thì các tiến trình không di chuyển từ một hàng đợi này tới hàng đợi khác vì các quá trình không thay đổi tính tự nhiên giữa giao tiếp và nền. Cách tổ chức có ích vì chi phí lập lịch thấp nhưng thiếu linh động và có thể dẫn đến tình trạng "đói CPU".

Tuy nhiên, **lập lịch hàng đợi phản hồi đa cấp** (multilevel feedback queue scheduling) cho phép một tiến trình di chuyển giữa các hàng đợi. Ý tưởng là tách riêng các tiến trình với các đặc điểm chu kỳ CPU khác nhau. Nếu một tiến trình dùng quá nhiều thời gian CPU thì nó sẽ được di chuyển tới hàng đợi có độ ưu tiên thấp. Cơ chế này để lại các tiến trình hướng nhập/xuất và các tiến trình giao tiếp trong các hàng đợi có độ ưu tiên cao hơn. Tương tự, một tiến trình chờ quá lâu trong hàng đợi có độ ưu

tiên thấp hơn có thể được di chuyển tới hàng đợi có độ ưu tiên cao hơn. Đây là hình thức của sự lão hóa nhằm ngăn chặn sự đói CPU.

Thí dụ, xét một bộ lập lịch hàng đợi phản hồi nhiều cấp với ba hàng đợi được đánh số từ 0 tới 2 (như hình 3.12). Bộ lập lịch trước tiên thực thi tất cả tiến trình chứa trong hàng đợi 0. Chỉ khi hàng đợi 0 rỗng nó sẽ thực thi các tiến trình trong hàng đợi 1. Tương tự, các tiến trình trong hàng đợi 2 sẽ được thực thi chỉ nếu hàng đợi 0 và 1 rỗng. Một tiến trình đến hàng đợi 1 sẽ ưu tiên hơn tiến trình đến hàng đợi 2. Tương tự, một tiến trình đến hàng đợi 0 sẽ ưu tiên hơn một tiến trình vào hàng đợi 1.



Hình 3.12. Các hàng đợi phản hồi nhiều cấp

Một tiến trình đưa vào hàng đợi sẵn sàng được đặt trong hàng đợi 0. Một quá trình trong hàng đợi 0 được cho một định mức thời gian là 8 mili giây. Nếu nó không kết thúc trong thời gian này thì nó sẽ di chuyển vào đuôi của hàng đợi 1. Nếu hàng đợi 0 rỗng thì tiến trình tại đầu của hàng đợi 1 được cho định mức thời gian là 16 mili giây. Nếu nó không hoàn thành thì nó bị chiếm CPU và được đặt vào hàng đợi 2. Các tiến trình trong hàng đợi 2 được chạy trên cơ sở FCFS chỉ khi hàng đợi 0 và 1 rỗng.

Giải thuật lập lịch này cho độ ưu tiên cao nhất tới bất cứ tiến trình nào với chu kỳ CPU 8 mili giây hay ít hơn. Một tiến trình như thế sẽ nhanh chóng nhận CPU, kết thúc chu kỳ CPU của nó và bỏ đi chu kỳ I/O kế tiếp của nó. Các tiến trình cần hơn 8 mili giây nhưng ít hơn 24 mili giây được phục vụ nhanh chóng mặc dù với độ ưu tiên thấp hơn các tiến trình ngắn hơn. Các tiến trình dài tự động rơi xuống hàng đợi 2 và được phục vụ trong thứ tự FCFS với bất cứ chu kỳ CPU còn lại từ hàng đợi 0 và 1.

Nói chung, một bộ lập lịch hàng đợi phản hồi nhiều cấp được định nghĩa bởi các tham số sau:

- Số lượng hàng đợi
- Giải thuật lập lịch cho mỗi hàng đợi
- Phương pháp được dùng để xác định khi nâng cấp một tiến trình tới hàng đợi có độ ưu tiên cao hơn.Phương pháp được dùng để xác định khi nào chuyển một tiến trình tới hàng đợi có độ ưu tiên thấp hơn.
- Phương pháp được dùng để xác định hàng đợi nào một tiến trình sẽ đi vào và khi nào tiến trình đó cần phục vụ.

Định nghĩa bộ lập lịch dùng hàng đợi phản hồi nhiều cấp trở thành giải thuật lập lịch CPU phổ biến nhất. Bộ lập lịch này có thể được cấu hình để thích hợp với hệ

thống xác định. Tuy nhiên, bộ lập lịch này cũng yêu cầu một vài phương tiện chọn lựa giá trị cho tất cả tham số để định nghĩa bộ lập lịch tốt nhất. Mặc dù một hàng đợi phản hồi nhiều cấp là cơ chế phổ biến nhất nhưng nó cũng là cơ chế phức tạp nhất.

VI.4. Lập lịch đa bộ xử lý

Phần trên thảo luận chúng ta tập trung vào những vấn đề lập lịch CPU trong một hệ thống với một bộ vi xử lý đơn. Nếu có nhiều CPU, vấn đề lập lịch tương ứng sẽ phức tạp hơn. Nhiều khả năng đã được thử nghiệm và như chúng ta đã thấy với lập lịch CPU đơn bộ xử lý, không có giải pháp tốt nhất. Trong phần sau đây, chúng ta sẽ thảo luận văn tắt một số vấn đề tập trung về lập lịch đa bộ xử lý. Chúng ta tập trung vào những hệ thống mà các bộ xử lý của nó được xác định (hay đồng nhất) trong thuật ngữ chức năng của chúng; bất cứ bộ xử lý nào sẵn có thì có thể được dùng để chạy bất tiến trình nào trong hàng đợi. Chúng ta cũng cho rằng truy xuất bộ nhớ là đồng nhất (uniform memory access-UMA). Chỉ những chương trình được biên dịch đối với tập hợp chỉ thị của bộ xử lý được cho mới có thể được chạy trên chính bộ xử lý đó.

Ngay cả trong một bộ đa xử lý đồng nhất đôi khi có một số giới hạn cho việc lập lịch. Xét một hệ thống với một thiết bị nhập/xuất được gán tới một đường bus riêng của một bộ xử lý. Các tiến trình muốn dùng thiết bị đó phải được lập lịch biểu để chạy trên bộ xử lý đó, ngược lại thiết bị đó là không sẵn dùng. Nếu nhiều bộ xử lý xác định sẵn dùng thì chia sẻ tải có thể xảy ra. Nó có thể cung cấp một hàng đợi riêng cho mỗi bộ xử lý. Tuy nhiên, trong trường hợp này, một bộ xử lý có thể rảnh với hàng đợi rỗng, trong khi bộ xử lý khác rất bận. Để ngăn chặn trường hợp này, chúng ta dùng một hàng đợi sẵn sàng chung. Tất cả tiến trình đi vào một hàng đợi và được lập lịch trên bất cứ bộ xử lý sẵn dùng nào.

Trong một cơ chế như thế, một trong hai tiếp cận lập lịch có thể được dùng. Trong tiếp cận thứ nhất, mỗi bộ xử lý lập lịch chính nó. Mỗi bộ xử lý xem xét hàng đợi sẵn sàng chung và chọn một tiến trình để thực thi. Nếu chúng ta có nhiều bộ xử lý cố gắng truy xuất và cập nhật một cấu trúc dữ liệu chung thì mỗi bộ xử lý phải được lập trình rất cẩn thận. Chúng ta phải đảm bảo rằng hai bộ xử lý không chọn cùng tiến trình và tiến trình đó không bị mất từ hàng đợi. Tiếp cận thứ hai tránh vấn đề này bằng cách đề cử một bộ xử lý như bộ lập lịch cho các tiến trình khác, do đó tạo ra cấu trúc chủ-tớ (master-slave).

Một vài hệ thống thực hiện cấu trúc này từng bước bằng cách tất cả quyết định lập lịch, xử lý nhập/xuất và các hoạt động hệ thống khác được quản lý bởi một bộ xử lý đơn-một server chủ. Các bộ xử lý khác chỉ thực thi mã người dùng. Đa xử lý không đối xứng (asymmetric multiprocessing) đơn giản hơn đa xử lý đối xứng (symmetric multiprocessing) vì chỉ một tiến trình truy xuất các cấu trúc dữ liệu hệ thống, làm giảm đi yêu cầu chia sẻ dữ liệu. Tuy nhiên, nó cũng không hiệu quả. Các tiến trình giới hạn nhập/xuất có thể gây thắt cổ chai (bottleneck) trên một CPU đang thực hiện tất cả các hoạt động. Điển hình, đa xử lý không đối xứng được cài đặt trước trong một hệ điều hành và sau đó được nâng cấp tới đa xử lý đối xứng khi hệ thống tiến triển.

VI.5. Lập lịch thời gian thực

Trong chương đầu chúng ta đã tìm hiểu tổng quan về hệ điều hành thời thực và thảo luận tầm quan trọng của nó. Ở đây, chúng ta tiếp tục thảo luận bằng cách mô tả các điều kiện thuận lợi lập lịch cần để hỗ trợ tính toán thời thực trong hệ thống máy tính đa mục đích.

Tính toán thời thực được chia thành hai loại: hệ thống thời thực cứng (hardware real-time systems) được yêu cầu để hoàn thành một tác vụ tới hạn trong lượng thời gian được đảm bảo. Thông thường, một tiến trình được đưa ra xem xét cùng với khai

báo lượng thời gian nó cần để hoàn thành hay thực hiện nhập/xuất. Sau đó, bộ định thời biểu nhận được tiến trình, đảm bảo rằng tiến trình sẽ hoàn thành đúng giờ hay từ chối yêu cầu khi không thể. Điều này được gọi là đặt trước tài nguyên (resource reservation). Để đảm bảo như thế đòi hỏi bộ lập lịch biết chính xác bao lâu mỗi loại chức năng hệ điều hành mất để thực hiện và do đó mỗi thao tác phải được đảm bảo để mất lượng thời gian tối đa. Một đảm bảo như thế là không thể trong hệ thống với lưu trữ phụ và bộ nhớ ảo vì các hệ thống con này gây ra sự biến đổi không thể tránh hay không thể thấy trước trong lượng thời gian thực thi một tiến trình xác định. Do đó, hệ thống thời thực cứng được hình thành từ nhiều phần mềm có mục đích đặc biệt chạy trên phần cứng tận hiến cho các tiến trình tới hạn, và thiếu chức năng đầy đủ của các máy tính và các hệ điều hành hiện đại.

Tính toán thời gian thực mềm (soft real-time computing) ít nghiêm khắc hơn. Nó yêu cầu các tiến trình tới hạn nhận độ ưu tiên cao hơn các tiến trình khác. Mặc dù thêm chức năng thời thực mềm tới hệ chia sẻ thời gian có thể gây ra việc cấp phát tài nguyên không công bằng và có thể dẫn tới việc trì hoãn lâu hơn hay thậm chí đói tài nguyên đối với một số tiến trình, nhưng nó ít có thể đạt được. Kết quả là hệ thống mục đích chung cũng có thể hỗ trợ đa phương tiện, đồ họa giao tiếp tốc độ cao, và nhiều tác vụ khác nhưng không hỗ trợ tính toán thời thực mềm.

Cài đặt chức năng thời thực mềm đòi hỏi thiết kế cẩn thận bộ lập lịch và các khía cạnh liên quan của hệ điều hành. Trước tiên, hệ thống phải có lập lịch trưng dụng và các tiến trình thời thực phải có độ ưu tiên cao nhất. Độ ưu tiên của các quá trình thời thực phải không giảm theo thời gian mặc dù độ ưu tiên của các tiến trình không thời thực có thể giảm. Thứ hai, độ trễ của việc điều phối phải nhỏ. Một quá trình thời thực nhỏ hơn, nhanh hơn có thể bắt đầu thực thi một khi nó có thể chạy.

Quản trị các thuộc tính đã được xem xét ở trên là tương đối đơn giản. Thí dụ, chúng ta có thể không cho phép một tiến trình hóa già trên các tiến trình thời thực, do đó đảm bảo rằng độ ưu tiên của các tiến trình không thay đổi. Tuy nhiên, đảm bảo thuộc tính sau đây phức tạp hơn. Vấn đề là nhiều hệ điều hành gồm hầu hết ấn bản của UNIX bị bắt buộc chờ lời gọi hệ thống hoàn thành hay nghẽn nhập/xuất xảy ra trước khi thực hiện chuyển ngữ cảnh. Độ trễ điều phối trong những hệ thống như thế có thể dài vì một số lời gọi hệ thống phức tạp và một vài thiết bị nhập/xuất chậm.

Để giữ độ trễ điều phối chậm, chúng ta cần cho phép các lời gọi hệ thống được trưng dụng. Có nhiều cách để đạt mục đích này. Cách thứ nhất là chèn các điểm trưng dụng (preemption points) trong những lời gọi hệ thống có khoảng thời gian dài, kiểm tra để thấy tiến trình ưu tiên cao cần được thực thi hay không. Nếu đúng, thì chuyển ngữ cảnh xảy ra và khi tiến trình có độ ưu tiên kết thúc, tiến trình bị ngắt tiếp tục với lời gọi hệ thống. Các điểm trưng dụng chỉ có thể được đặt tại vị trí "an toàn" trong nhân- nơi mà những cấu trúc dữ liệu hiện tại không được cập nhật. Ngay cả với độ trễ điều phối trưng dụng có thể lớn vì chỉ một vài điểm trưng dụng có thể được thêm vào nhân trong thực tế.

Một phương pháp khác để giải quyết sự trưng dụng là làm toàn bộ nhân có thể trưng dụng. Để đảm bảo các hoạt động thực hiện đúng, tất cả cấu trúc dữ liệu nhân phải được bảo vệ thông qua việc sử dụng các cơ chế đồng bộ hóa. Với phương pháp này, nhân luôn có thể trưng dụng vì bất cứ dữ liệu nhân được cập nhật được bảo vệ từ việc sửa đổi bởi tiến trình có độ ưu tiên cao. Đây là một phương pháp hiệu quả nhất trong việc sử dụng rộng rãi; nó được dùng trong Solaris 2.

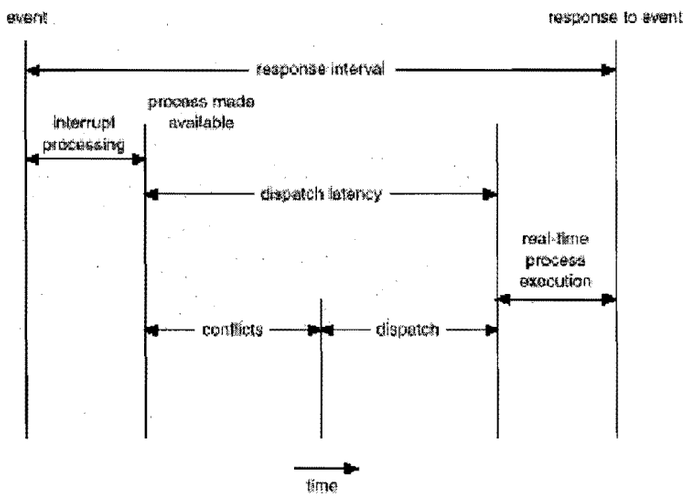
Nhưng điều gì xảy ra nếu tiến trình có độ ưu tiên cao cần đọc hay sửa đổi dữ liệu nhân hiện đang được truy xuất bởi tiến trình khác có độ ưu tiên thấp hơn? Quá trình có độ ưu tiên cao đang chờ tiến trình có độ ưu tiên thấp kết thúc. Trường hợp này

được gọi là đảo ngược độ ưu tiên (prioprity inversion). Thật vậy, một chuỗi các tiến trình đang truy xuất tài nguyên mà tiến trình có độ ưu tiên cao cần. Vấn đề này có thể giải quyết bằng giao thức kế thừa độ ưu tiên (priority-inheritance protocol) trong đó tất cả tiến trình này (các tiến trình này truy xuất tài nguyên mà tiến trình có độ ưu tiên cao cần) kế thừa độ ưu tiên cao cho đến khi chúng được thực hiện với tài nguyên trong câu hỏi. Khi chúng kết thúc, độ ưu tiên của chúng chuyển trở lại giá trị ban đầu của nó.

Trong hình 3.13, chúng ta hiển thị sự thay đổi của độ trễ điều phối. Giai đoạn xung đột (conflict phase) của độ trễ điều phối có hai thành phần:

- Sự trưng dụng bất cứ tiến trình nào đang chạy trong nhân
- Giải phóng tài nguyên các tiến trình có độ ưu tiên thấp được yêu cầu bởi tiến trình có độ ưu tiên cao

Thí dụ, trong Solaris 2 độ trễ điều phối với sự trưng dụng bị vô hiệu hóa khi vượt qua 100 mili giây. Tuy nhiên, độ trễ điều phối với sự trưng dụng được cho phép thường được giảm xuống tới 2 mili giây.



Hình 3.13. Độ trễ gửi

VI.6. Đánh giá giải thuật

Chúng ta chọn một giải thuật lập lịch CPU cho một hệ thống xác định như thế nào? Có nhiều giải thuật lập lịch, mỗi giải thuật với các tham số của riêng nó. Do đó, chọn một giải thuật có thể là khó. Vấn đề đầu tiên là định nghĩa các tiêu chuẩn được dùng trong việc chọn một giải thuật. Các tiêu chuẩn thường được định nghĩa trong thuật ngữ khả năng sử dụng CPU, thời gian đáp ứng hay thông lượng. Để chọn một giải thuật, trước hết chúng ta phải định nghĩa trọng số quan trọng của các thước đo này. Tiêu chuẩn của chúng ta có thể gồm các thước đo như:

- Khả năng sử dụng CPU tối đa dưới sự ràng buộc thời gian đáp ứng tối đa là 1 giây.
- Thông lượng tối đa như thời gian hoàn thành (trung bình) tỉ lệ tuyến tính với tổng số thời gian thực thi.

Một khi các tiêu chuẩn chọn lựa được định nghĩa, chúng ta muốn đánh giá các giải thuật khác nhau dưới sự xem xét. Chúng ta mô tả các phương pháp đánh giá khác nhau trong những phần dưới đây

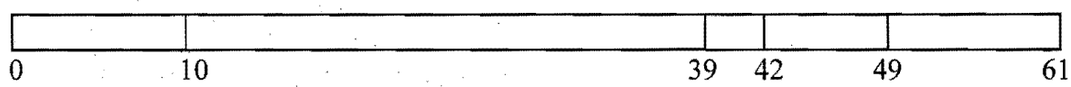
VI.6.1. Mô hình quyết định

Một loại quan trọng của phương pháp đánh giá được gọi là đánh giá phân tích (analytic evaluation). Đánh giá phân tích dùng giải thuật được cho và tải công việc hệ thống để tạo ra công thức hay số đánh giá năng lực của giải thuật cho tải công việc đó.

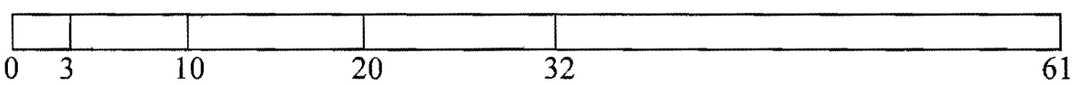
Một dạng đánh giá phân tích là **mô hình xác định** (deterministic modeling). Phương pháp này lấy tải công việc đặc biệt được xác định trước và định nghĩa năng lực của mỗi giải thuật cho tải công việc đó. Thí dụ, giả sử rằng chúng ta có tải công việc được hiển thị trong bảng dưới. Tất cả 5 tiến trình đến tại thời điểm 0 trong thứ tự được cho, với chiều dài của thời gian chu kỳ CPU được tính bằng mili giây.

Tiến trình	Thời gian xử lý
P1	10
P2	29
P3	3
P4	7
P5	12

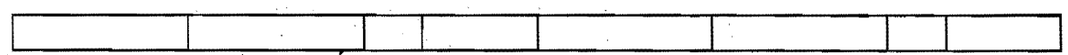
Xét giải thuật lập lịch FCFS, SJF và RR (định mức thời gian=10 mili giây) cho tập hợp tiến trình này. Giải thuật nào sẽ cho thời gian chờ đợi trung bình tối thiểu? Đối với giải thuật FCFS, chúng ta sẽ thực thi các tiến trình này như sau:



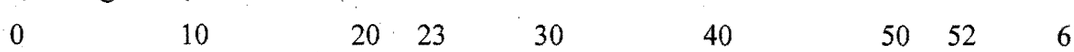
Thời gian chờ đợi là 0 mili giây cho tiến trình P_1, 32 mili giây cho tiến trình P_2, 39 giây cho tiến trình P_3, 42 giây cho tiến trình P_4 và 49 mili giây cho tiến trình P_5. Do đó, thời gian chờ đợi trung bình là (0 + 10 + 39 + 42 + 49)/5= 28 mili giây. Với lập lịch không trưng dụng SJF, chúng ta thực thi các tiến trình như sau:



Thời gian chờ đợi là 10 mili giây cho tiến trình P_1, 32 mili giây cho tiến trình P_2, 0 mili giây cho tiến trình P_3, 3 mili giây cho tiến trình P_4, và 20 giây cho tiến trình P_5. Do đó, thời gian chờ đợi trung bình là (10 + 32 + 0 + 3 + 20)/5= 13 mili giây. Với giải thuật



RR, chúng ta thực thi các tiến trình như sau:



Thời gian chờ đợi là 0 mili giây cho tiến trình P_1, 32 mili giây cho tiến trình P_2, 20 mili giây cho tiến trình P_3, 23 mili giây cho tiến trình P_4, và 40 mili giây cho tiến trình P_5. Do đó, thời gian chờ đợi trung bình là (0 + 32 + 20 + 23 + 40)/5 = 23 mili giây.

Trong trường hợp này, chúng ta thấy rằng, chính sách SJF cho kết quả ít hơn ½ thời gian chờ đợi trung bình đạt được với giải thuật FCFS; giải thuật RR cho chúng ta giá trị trung gian. Mô hình xác định là đơn giản và nhanh. Nó cho các con số chính xác, cho phép các giải thuật được so sánh với nhau. Tuy nhiên, nó đòi hỏi các số đầu vào chính xác và các trả lời của nó chỉ áp dụng cho những trường hợp đó. Việc dùng chủ yếu của mô hình xác định là mô tả giải thuật lập lịch và cung cấp các thí dụ. Trong các

trường hợp, chúng ta đang chạy cùng các chương trình lặp đi lặp lại và có thể đo các yêu cầu xử lý của chương trình một cách chính xác, chúng ta có thể dùng mô hình xác định để chọn giải thuật lập lịch. Qua tập hợp các thí dụ, mô hình xác định có thể hiển thị khuynh hướng được phân tích và chứng minh riêng. Thí dụ, có thể chứng minh rằng đối với môi trường được mô tả (tất cả tiến trình và thời gian của chúng sẵn dùng tại thời điểm 0), chính sách SJF sẽ luôn cho kết quả thời gian chờ đợi là nhỏ nhất.

Tuy nhiên, nhìn chung mô hình xác định quá cụ thể và yêu cầu tri thức quá chính xác để sử dụng nó một cách có ích.

VI.6.2. Mô hình hàng đợi

Các tiến trình được chạy trên nhiều hệ thống khác nhau từ ngày này sang ngày khác vì thế không có tập hợp tiến trình tĩnh (và thời gian) để dùng cho mô hình xác định. Tuy nhiên, những gì có thể được xác định là sự phân bổ chu kỳ CPU và I/O. Sự phân bổ này có thể được đo và sau đó được tính xấp xỉ hay ước lượng đơn giản. Kết quả là một công thức toán mô tả xác suất của một chu kỳ CPU cụ thể. Thông thường, sự phân bổ này là hàm mũ và được mô tả bởi giá trị trung bình của nó. Tương tự, sự phân bổ thời gian khi các tiến trình đến trong hệ thống-phân bổ thời gian đến-phải được cho.

Hệ thống máy tính được mô tả như một mạng các server. Mỗi server có một hàng đợi cho các tiến trình. CPU là một server với hàng đợi sẵn sàng của nó, như là một hệ thống nhập/xuất với các hàng đợi thiết bị. Biết tốc độ đến và tốc độ phục vụ, chúng ta có thể tính khả năng sử dụng, chiều dài hàng đợi trung bình, thời gian chờ trung bình,..Lĩnh vực nghiên cứu này được gọi là phân tích mạng hàng đợi (queueing-network analysis).

Thí dụ, gọi n là chiều dài hàng đợi trung bình (ngoại trừ các tiến trình đang được phục vụ), gọi W là thời gian chờ đợi trung bình trong hàng đợi và λ là tốc độ đến trung bình cho các tiến trình mới trong hàng đợi (chẳng hạn 3 tiến trình trên giây). Sau đó, chúng ta mong đợi trong suốt thời gian W một tiến trình chờ, $\lambda \times W$ các tiến trình mới sẽ đến trong hàng đợi. Nếu hệ thống ở trong trạng thái đều đặn thì số lượng quá trình rời hàng đợi phải bằng số lượng tiến trình đến. Do đó,

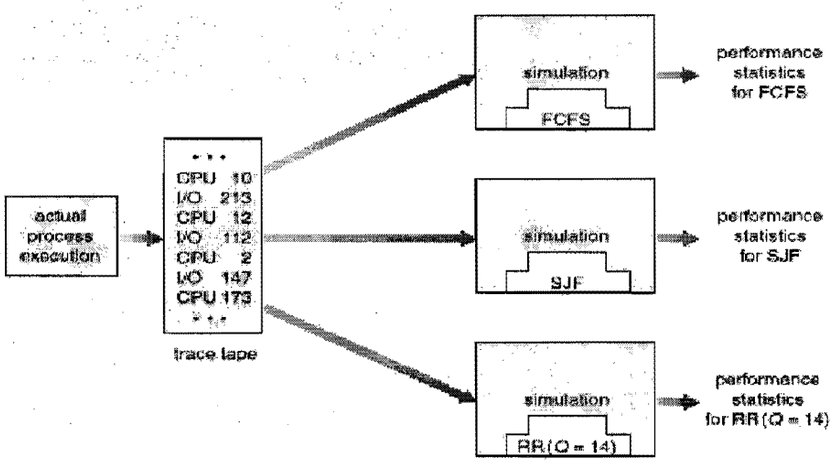
$$n = \lambda \times W$$

Công thức này được gọi là công thức Little. Công thức Little là đặc biệt có ích vì nó phù hợp cho bất cứ giải thuật lập lịch và sự phân bổ các tiến trình đến. Chúng ta sử dụng công thức Little để tính một trong ba biến, nếu chúng ta biếthai biến khác. Thí dụ, nếu chúng ta biết có 7 tiến trình đến mỗi giây (trung bình) và thường có 14 tiến trình trong hàng đợi thì chúng ta có thể tính thời gian chờ đợi trungbình trên mỗi tiến trình là 2 giây.

Phân tích hàng đợi có thể có ích trong việc so sánh các giải thuật lập lịch nhưng nó cũng có một số giới hạn. Hiện nay, các loại giải thuật và sự phân bổ được quản lý là tương đối giới hạn. Tính toán của các giải thuật phức tạp và sự phân bổ là rất khó để thực hiện. Do đó, phân bổ đến và phục vụ thường được định nghĩa không thực tế, nhưng dễ hướng dẫn về mặt tính toán. Thông thường cần thực hiện một số giả định độc lập có thể không chính xác. Do đó, để chúng sẽ có thể tính câu trả lời, các mô hình hàng đợi thường chỉ xấp xỉ hệ thống thật. Vì thế, độ chính xác của các kết quả tính toán có thể là sự nghi vấn.

VI.6.3. Mô phỏng

Để đạt được sự đánh giá các giải thuật lập lịch chính xác hơn, chúng ta có thể dùng mô phỏng (simulations). Mô phỏng liên quan đến lập trình một mô hình hệ thống máy tính. Cấu trúc dữ liệu phần mềm biểu diễn các thành phần quan trọng của hệ thống. Bộ mô phỏng có một biến biểu diễn đồng hồ; khi giá trị của biến này tăng, bộ mô phỏng sửa đổi trạng thái hệ thống để phản ánh các hoạt động của các thiết bị, các tiến trình và các bộ lập lịch. Khi sự mô phỏng thực thi, các thống kê hiển thị năng lực của giải thuật được tập hợp và in ra.



Hình 3.14. Đánh giá các bộ lập lịch CPU băng mô phỏng

Dữ liệu để định hướng sự mô phỏng có thể được sinh ra trong nhiều cách. Cách thông dụng nhất dùng bộ sinh số ngẫu nhiên, được lập trình để sinh ra các quá trình, thời gian chu kỳ CPU, đến, đi của tiến trình,..dựa trên phân bố xác suất. Sự phân bố này có thể được định nghĩa dạng toán học (đồng nhất, hàm mũ, phân bố Poisson) hay theo kinh nghiệm. Nếu sự phân bố được định nghĩa theo kinh nghiệm thì các thước đo của hệ thống thật dưới sự nghiên cứu là lấy được. Các kết quả được dùng để định nghĩa sự phân bố thật sự các sự kiện trong hệ thống thực và sau đó sự phân bố này có thể được dùng để định hướng việc mô phỏng.

Tuy nhiên, một mô phỏng hướng phân bố có thể không chính xác do mối quan hệ giữa các sự kiện tiếp theo trong hệ thống thực. Sự phân bố thường xuyên hiển thị chỉ bao nhiêu sự kiện xảy ra; nó không hiển thị bất cứ thứ gì về thứ tự xảy ra của chúng. Để sửa chữa vấn đề này, chúng ta dùng băng từ ghi vết (trace tapes). Chúng ta tạo một băng từ ghi vết bằng cách giám sát hệ thống thực, ghi lại chuỗi các sự kiện thật (như hình 3.14.). Sau đó, thứ tự này được dùng để định hướng việc mô phỏng.

Băng từ ghi vết cung cấp cách tuyệt vời để so sánh chính xác hai giải thuật trên cùng một tập hợp dữ liệu vào thật. Phương pháp này có thể cung cấp các kết quả chính xác cho dữ liệu vào của nó.

Tuy nhiên, mô phỏng có thể rất đắt và thường đòi hỏi hàng giờ máy tính để thực hiện. Một mô phỏng chi tiết hơn cung cấp các kết quả chính xác hơn nhưng cũng yêu cầu nhiều thời gian máy tính hơn. Ngoài ra, các băng từ ghi vết có thể yêu cầu lượng lớn không gian lưu trữ. Cuối cùng, thiết kế, mã, gỡ rối của bộ mô phỏng là một tác vụ quan trọng.

VI.6.4. Cài đặt

Ngay cả mô phỏng cũng cho độ chính xác có giới hạn. Chỉ có cách chính xác hoàn toàn để đánh giá giải thuật lập lịch là mã hóa (code) nó, đặt nó vào trong hệ điều

hành và xem nó làm việc như thế nào. Tiếp cận này đặt một giải thuật thật sự vào hệ thống thật để đánh giá dưới điều kiện hoạt động thật sự.

Khó khăn chủ yếu là chi phí của tiếp cận. Chi phí bao gồm không chỉ mã hóa giải thuật và sửa đổi hệ điều hành để hỗ trợ nó cũng như các cấu trúc dữ liệu được yêu cầu mà còn phản ứng của người dùng đối với sự thay đổi liên tục hệ điều hành. Hầu hết người dùng không quan tâm việc xây dựng một hệ điều hành tốt hơn; họ chỉ đơn thuần muốn biết các tiến trình của họ thực thi và dùng các kết quả của chúng. Một hệ điều hành thay đổi liên tục không giúp cho người dùng nhận thấy công việc của họ được thực hiện. Một dạng của phương pháp này được dùng phổ biến cho việc cài đặt máy tính mới. Thí dụ, một tiện ích Web mới có thể mô phỏng tải người dùng được phát sinh trước khi nó "sống" (goes live), để xác định bất cứ hiện tượng thắt cổ chai trong tiện ích và để ước lượng bao nhiêu người dùng hệ thống có thể hỗ trợ.

Một khó khăn khác với bất cứ việc đánh giá giải thuật nào là môi trường trong đó giải thuật được dùng sẽ thay đổi. Môi trường sẽ thay đổi không chỉ trong cách thông thường như những chương trình mới được viết và các loại vấn đề thay đổi, mà còn kết quả năng lực của bộ lập lịch. Nếu các tiến trình được cho với độ ưu tiên ngắn thì người dùng có thể tách các tiến trình lớn thành tập hợp các tiến trình nhỏ hơn. Nếu tiến trình giao tiếp được cho độ ưu tiên vượt qua các tiến trình không giao tiếp thì người dùng có thể chuyển tới việc dùng giao tiếp.

Thí dụ, trong DEC TOPS-20, hệ thống được phân loại các tiến trình giao tiếp và không giao tiếp một cách tự động bằng cách xem lượng nhập/xuất thiết bị đầu cuối. Nếu một tiến trình không có nhập hay xuất tới thiết bị đầu cuối trong khoảng thời gian 1 phút thì tiến trình được phân loại là không giao tiếp và được di chuyển tới hàng đợi có độ ưu tiên thấp. Chính sách này dẫn đến trường hợp một người lập trình sửa đổi chương trình của mình để viết một ký tự bất kỳ tới thiết bị đầu cuối tại khoảng thời gian đều đặn ít hơn 1 phút. Hệ thống này cho những chương trình này có độ ưu tiên cao mặc dù dữ liệu xuất của thiết bị đầu cuối là hoàn toàn không có ý nghĩa.

Các giải thuật có khả năng mềm dẻo nhất có thể được thay đổi bởi người quản lý hayngười dùng. Trong suốt thời gian xây dựng hệ điều hành, thời gian khởi động, thời gian chạy, các biến được dùng bởi các bộ lập lịch có thể được thay đổi để phản ánh việc sử dụng của hệ thống trong tương lai. Yêu cầu cho việc lập lịch mềm dẻo là một trường hợp khác mà ở đó sự tách riêng các cơ chế từ chính sách là có ích. Thí dụ, nếu các hóa đơn cần được xử lý và in lập tức nhưng thường được thực hiện như công việc lô có độ ưu tiên thấp, hàng đợi lô được cho tạm thời độ ưu tiên cao hơn. Tuy nhiên, rất ít hệ điều hành chấp nhận loại lập lịch này.

VI.7. Ghi nhớ

Lập lịch CPU là một tác vụ chọn một tiến trình đang chờ từ hàng đợi sẵn sàng và cấp phát CPU tới nó. CPU được cấp phát tới tiến trình được chọn bởi bộ cấp phát. Lập lịch đến trước, được phục vụ trước (FCFS) là giải thuật lập lịch đơn giản nhất, nhưng nó có thể gây các tiến trình ngắn chờ các tiến trình tiến trình quá dài.

Lập lịch ngắn nhất, phục vụ trước (SJF) có thể tối ưu, cung cấp thời gian chờ đợi trung bình ngắn nhất. Cài đặt lập lịch SJF là khó vì đoán trước chiều dài của chu kỳ CPU kế tiếp là khó. Giải thuật SJF là trường hợp đặc biệt của giải thuật lập lịch trưng dụng thông thường. Nó đơn giản cấp phát CPU tới tiến trình có độ ưu tiên cao nhất. Cả hai lập lịch độ ưu tiên và SJF có thể gặp phải trở ngại của việc đói tài nguyên.

Lập lịch quay vòng (RR) là hợp lý hơn cho hệ thống chia sẻ thời gian. Định thời RR cấp phát CPU tới tiến trình đầu tiên trong hàng đợi sẵn sàng cho q đơn vị thời gian, ở đây q là định mức thời gian. Sau q đơn vị thời gian, nếu tiến trình này không trả lại

CPU thì nó bị chiếm và tiến trình này được đặt vào đuôi của hàng đợi sẵn sàng. Vấn đề quan trọng là chọn định mức thời gian. Nếu định mức quá lớn, thì lập lịch RR giảm hơn lập lịch FCFS ; nếu định mức quá nhỏ thì chi phí lập lịch trong dạng thời gian chuyển ngữ cảnh trở nên thừa.

Giải thuật FCFS là không ưu tiên; giải thuật RR là ưu tiên. Các giải thuật SJF và ưu tiên có thể ưu tiên hoặc không ưu tiên.

Các giải thuật hàng đợi nhiều cấp cho phép các giải thuật khác nhau được dùng cho các loại khác nhau của tiến trình. Chung nhất là hàng đợi giao tiếp ở chế độ hiển thị dùng lập lịch RR và hàng đợi lô chạy ở chế độ nền dùng lập lịch FCFS.

Hàng đợi phản hồi nhiều cấp cho phép các tiến trình di chuyển từ hàng đợi này sang hàng đợi khác.

Vì có nhiều giải thuật lập lịch sẵn dùng, chúng ta cần các phương pháp để chọn giữa chúng. Các phương pháp phân tích dùng cách thức phân tích toán học để xác định năng lực của giải thuật. Các phương pháp mô phỏng xác định năng lực bằng cách phỏng theo giải thuật lập lịch trên những mẫu 'đại diện' của tiến trình và tính năng lực kết quả.