

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

**KHOA CÔNG NGHỆ THÔNG TIN 1**



****

**BÀI TẬP LỚN**

**BỘ MÔN:** HỆ ĐIỀU HÀNH

**CHUYÊN ĐỀ:** LẬP LỊCH HÀNG ĐỢI PHẢN HỒI ĐA CẤP-MLFQ

**NHÓM MÔN HỌC:** 02

**Giảng viên:** Nguyễn Văn Thủy

**Sinh viên:** Đặng Thị Vân Anh

**Mã sinh viên:** B19DCCN010

***Hà Nội – 2021***

**MỞ ĐẦU**

Đề tài này nghiên cứu cách giải quyết vấn đề đã và đang phát triển, một trong các cách tiếp cận nổi tiếng nhất để lập lịch, được gọi là hàng đợi phản hồi đa cấp – The Multi-Level Feedback Queue (MLFQ). Lập lịch với hàng đợi phản hồi đa cấp được mô tả lần đầu tiên bởi Corbato et al. vào năm 1962, trong một hệ thống được gọi là Hệ thống chia sẻ thời gian tương thích – Compatible Time-Sharing System (CTSS), và công trình này, cùng với các công trình khác về Đa phương tiện, đã khiến ACM trao cho Corbató giải thưởng cao nhất, giải Turing. Bộ lập lịch này đã được cải thiện trong suốt nhiều năm sau đó, tới các triển khai mà ta sẽ gặp được trong một số hệ thống hiện đại.

Hàng đợi phản hồi đa cấp là một ví dụ tuyệt vời về hệ thống mà từ các hành vi trong quá khứ để dự đoán cho hành vi trong tương lai. Cách tiếp cận như thế là phổ biến trong các hệ điều hành (và nhiều phần khác trong Khoa học máy tính, bao gồm các bộ dự đoán nhánh phần cứng và các thuật toán bộ nhớ đệm). Các cách tiếp cận như vậy hoạt động khi các công việc bao gồm nhiều giai đoạn về hành vi và từ đó có thể dự đoán trước được hành vi nào sẽ xảy ra tiếp theo. Tất nhiên, phải cẩn thận với những kỹ thuật như vậy, vì chúng có thể dễ dàng xảy ra sai sót và khiến hệ thống đưa ra những quyết định tồi tệ hơn khi chúng không có thông tin gì về công việc cả.

**MỤC LỤC**

Mục lục

[**ĐẶT VẤN ĐỀ** 1](#_Toc87791868)

[**CÁCH GIẢI QUYẾT** 2](#_Toc87791869)

[**1.** **MLFQ: Các quy tắc cơ bản** 2](#_Toc87791870)

[**2.** **Cách thay đổi thứ tự mức ưu tiên** 3](#_Toc87791871)

[**3.** **Sự tăng cường ưu tiên** 8](#_Toc87791872)

[**4.** **Cải thiện việc giám sát hoạt động** 9](#_Toc87791873)

[**5.** **Điều chỉnh MLFQ và những vấn đề khác** 10](#_Toc87791874)

[**6.** **MLFQ: Tổng kết** 12](#_Toc87791875)

[**KẾT QUẢ CHẠY CHƯƠNG TRÌNH** 14](#_Toc87791876)

[**PHẦN MỞ RỘNG:** 15](#_Toc87791877)

[Lập lịch cho thời hạn sớm nhất trước tiên– Earliest Deadline First (EDF) 15](#_Toc87791878)

# ĐẶT VẤN ĐỀ

Vấn đề cơ bản mà MLFQ cố gắng giải quyết được chia thành hai vấn chính. Đầu tiên, nó muốn tối ưu hóa thời gian quay vòng, được thực hiện bằng cách chạy các công việc ngắn hơn trước. Nhưng thật không may, các hệ điều hành nói chung không biết được một công việc sẽ chạy trong bao lâu, hay thông tin chính xác mà các thuật toán như SJF (hay STCF) yêu cầu. Thứ hai, MLFQ muốn tạo ra một hệ thống cảm ứng với người dùng tương tác (ví dụ như người dùng ngồi và nhìn vào màn hình, đợi đến khi một quá trình kết thúc), các thuật toán như Round Robin làm giảm thời gian phản ứng nhưng lại rất tệ cho thời gian quay vòng. Vì vậy, vấn đề ở đây chỉ ra rằng ta không biết trước thời gian chạy của một tiến trình (process), làm thế nào để xây dựng một lịch trình để đạt được những mục tiêu trên? Làm thế nào bộ lập lịch có thể đọc được, những thứ như hệ thống có đang chạy, các thuộc tính của công việc mà nó đang chạy, và từ đó đưa ra những quyết định lập lịch tốt hơn?

Điểm mấu chốt ở đây là: Làm thế nào chúng ta có thể thiết kế một lịch trình mà vừa giảm thiểu thời gian phản ứng cho việc làm tương tác, vừa giảm thiểu thời gian xoay vòng trong khi ta không biết về độ dài công việc được đặt ra.

# CÁCH GIẢI QUYẾT

1. **MLFQ: Các quy tắc cơ bản**

Để xây dựng một bộ lập lịch như vậy, ta sẽ mô tả những thuật toán cơ bản của hàng đợi phản hồi đa cấp; mặc dù các chi tiết cụ thể của các MLFQ được triển khai khác nhau, nhưng hầu hết các cách tiếp cận đều giống nhau.

Trong cách xử lý này, MLFQ có một số hàng đợi riêng biệt, mỗi mội hàng đợi được chỉ định một mức độ ưu tiên khác nhau. Tại bất kỳ thời điểm nào, một công việc sẵn sàng để chạy là trên một hàng đợi duy nhất. MLFQ sử dụng các mức độ ưu tiên để quyết định công việc nào nên chạy vào một thời điểm nhất định: một công việc có mức độ ưu tiên cao hơn (tức là công việc nằm ở đầu hàng đợi) được chọn để chạy.

Và tất nhiên, nhiều công việc có thể nằm trên một hàng đợi nhất định, do đó chúng có cùng một mức độ ưu tiên. Trong trường hợp này, chúng ta sẽ chỉ sử dụng Round Robin cho các công việc đó.

Từ đó, ta đi đến hai quy tắc cơ bản đầu tiên của MLFQ:

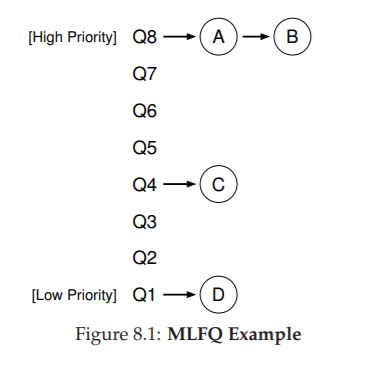
* Quy tắc 1: Nếu mức ưu tiên(A) > mức ưu tiên(B), A chạy (B không chạy)
* Quy tắc 2: Nếu mức ưu tiên(A) = mức ưu tiên(B), A và B cùng chạy bằng giải thuật Roind Robin

Chìa khóa cho việc lập lịch MLFQ là nằm ở cách sắp đặt các mức độ ưu tiên của bộ lập lịch. Thay vì đưa ra một mức độ ưu tiên cố định cho từng công việc, MLFQ thay đổi mức độ ưu tiên của một công việc dựa trên việc *quan sát hành vi* của nó.

Ví dụ, nếu một công việc liên tục thoát khỏi CPU trong khi chờ nhập dữ liệu từ bàn phím, MLFQ sẽ giữ mức ưu tiên cao, vì đây là cách chương trình có thể hoạt động. Thay vào đó, nếu một công việc sử dụng CPU nhiều trong một khoảng thời gian dài, MLFQ sẽ giảm mức độ ưu tiên của nó. Bằng cách này, MLFQ sẽ cố gắng tìm hiểu về các quy trình khi chúng chạy và từ đó sử dụng lịch sử công việc để dự đoán hành vi tiếp theo của nó.

Nếu đưa ra một hình vẽ mô tả những gì hàng đợi sẽ trông như thế nào tại một thời điểm xác định, ta có thể nhận thấy trong hình sau (Hình 8.1). Trong hình, hai công việc (A và B) ở mức độ ưu tiên cao nhất, trong khi công việc C ở giữa và công việc D ở mức thấp nhất. Cách MLFQ hoạt động hiện tại, bộ lập lịch chỉ thay thế thời gian giữa A và B vì chúng là những công việc có mức độ ưu tiên cao nhất trong hệ thống. Và công việc C và D, sẽ không bao giờ được chạy.

Những gì cần hiểu được ở đây là cách thức mà mức độ ưu tiên của công việc *thay đổi* theo thời gian.



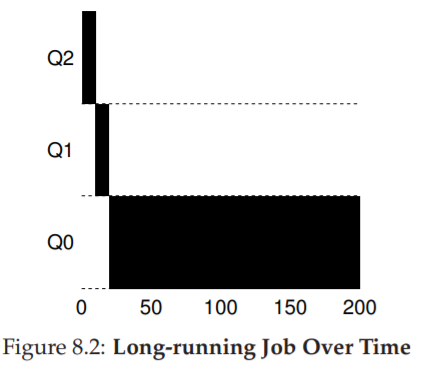
1. **Cách thay đổi thứ tự mức ưu tiên**

Bây giờ, sẽ phải quyết định MLFQ sẽ phải thay đổi mức độ ưu tiên của một công việc như thế nào (và nó nằm trong hàng đợi nào) trong suốt thời gian hoàn thành công việc. Để làm được diều này, phải ghi nhớ khối lượng công việc của mình: sự kết hợp của việc tương tác giữa các công việc ngắn hơn đang chạy ( và có thể thường xuyên ngừng hoạt động của CPU), và một số các công việc “ràng buộc CPU” lâu hơn, nó cần nhiều thời gian của CPU nhưng thời gian phản hồi lại không mấy quan trọng. Dưới đây là các quy tắc tiếp theo cho cách thay đổi thứ tự ưu tiên của các công việc đối với thuật toán điều chỉnh ưu tiên:

* Quy tắc 3: Khi một công việc được đưa vào hệ thống, nó được đặt ở vị trí ưu tiên cao nhất (hàng đợi trên cùng).
* Quy tắc 4a: Nếu một công việc sử dụng hết một lát cắt thời gian trong khi chạy, mức độ ưu tiên của nó sẽ bị giảm xuống (tức là nó di chuyển xuống một hàng đợi).
* Quy tắc 4b: Nếu một công việc thoát khỏi CPU trước khi lát cắt thời thời kết thúc, nó vẫn ở cùng một mức độ ưu tiên.

**Ví dụ 1: Một công việc dài hạn duy nhất**

Xét một số ví dụ. Đầu tiên, ta sẽ xem xét điều gì đó sẽ xảy ra khi đã có một công việc đang hoạt động lâu dài bên trong hệ thống. Hình 8.2 sẽ cho thấy điều gì sẽ xảy ra đối với công việc vượt quá thời gian này trong bộ lập lịch với ba hàng đợi.

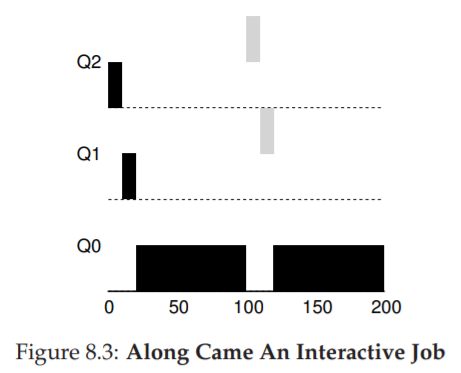


Như có thể thấy trong ví dụ trên, công việc được ưu tiên cao nhất (Q2). Sau lát cắt thời gian 10ms, bộ lập lịch đã giảm bớt mức độ ưu tiên của từng công việc một, và sau đó, công việc nằm ở Q1. Sau khi chạy ở Q1 một lát cắt thời gian, công việc cuối cùng được hạ xuống mức ưu tiên thấp nhất trong hệ thống (Q0), và tiếp tục chạy.

**Ví dụ 2: Cùng đến với một công việc ngắn hạn**

Bây giờ chúng ta hãy xem xét một ví dụ phức tạp hơn. Trong ví dụ này có hai công việc: A là công việc đòi hỏi CPU phải chạy lâu dài và B là một công việc tương tác ngắn. Giả sử A đã chạy một thời gian và sau đó B đến. Chuyện gì sẽ xảy ra?

Hình 8.3 Là sơ đồ thể hiện kết quả của giả sử trên. A (được hiển thị bằng màu đen) đang chạy dọc theo hàng đợi có mức độ ưu tiên thấp nhất (giống như bất kỳ công việc khác đòi hỏi nhiều từ CPU), B (được hiển thị bằng màu xám) đến thời điểm tại T=100, được đưa vào hàng đợi cao nhất, vì thời gian chạy của nó ngắn (chỉ tốn 20ms), B hoàn thành trong hai lát cắt thời gian trước khi được chuyển xuống hàng đợi cuối cùng; sau đó A tiếp tục chạy (ở mức ưu tiên thấp).

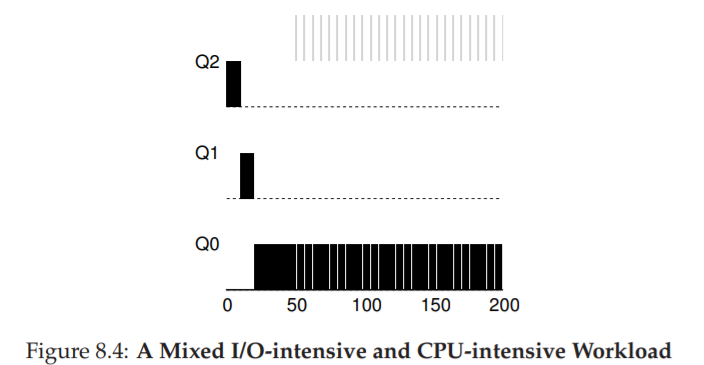


Từ ví dụ này, có thể chỉ ra một trong những mục tiêu của thuật toán: vì nó không biết liệu công việc được đưa vào hệ thống là ngắn hạn hay dài hạn, trước tiên nó phải giả định rằng đó có thể là công việc ngắn hạn, vì thế mức độ ưu tiên của công việc đó là cao. Nếu nó thực sự là công việc ngắn hạn, nó sẽ chạy một cách nhanh chóng và hoàn thành. Nếu nó không phải công việc ngắn hạn, nó sẽ từ từ di chuyển xuống hàng đợi, và do đó nó sẽ sớm chứng tỏ bản thân là công việc dài hạn hơn là giống như hàng loạt các quá trình khác.

**Ví dụ 3: Thế còn với I/O?**

Bây giờ ta sẽ xem xét một ví dụ với một số I/O. Theo như quy tắc 4a nêu trên, nếu một quy trình từ bỏ bộ xử lý trước khi lát cắt thời gian của nó đã sử dụng hết, ta giữ nó ở cùng mức độ ưu tiên. Mục đích của quy tắc này rất đơn giản: ví dụ, nếu một công việc tương tác đang thực hiện rất nhiều I/O (giả sử chờ người dùng nhập dữ liệu từ ban phím hoặc chuột), nó sẽ rời khỏi CPU trước khi lát cắt thời gian của nó kết thúc; trong trường hợp như vậy, ta nó ở cùng một mức độ.

Hình 8.4 cho ta thấy một ví dụ về cách hoạt động của điều này, với một công việc tương tác B (hiển thị bằng màu xám) chỉ cần CPU trong 1ms trước khi thực hiện cạnh tranh I/O cho CPU với một loạt công việc dài hạn A đang chạy (được hiển thị bằng màu đen). Cách MLFQ giữ B ở mức ưu tiên cao nhất vì B sẽ tiếp tục giải phóng CPU; nếu B chỉ là một công việc tương tác, MLFQ còn đạt được mục tiêu của nó là chạy các công việc tương tác một cách nhanh chóng.



**Các vấn đề hiện tại với MLFQ**

Cho đến hiện tại, ta có khái niệm cơ bản về một MLFQ. Nó dường như làm khá tốt công việc, chia sẻ CPU khá ổn định giữa các công việc dài hạn , ngắn hạn và cho phép I/O những công việc tương tác chuyên sâu chạy nhanh chóng. Nhưng thật không may, vẫn còn những sai sót nghiêm trọng.

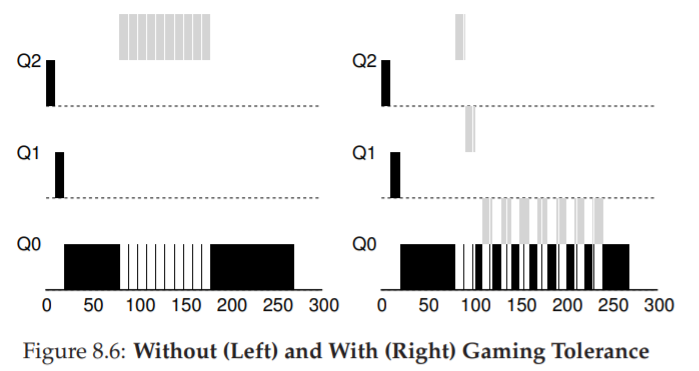
Đầu tiên, đó là vấn đề “đói”: nếu có “quá nhiều” công việc tương tác trong hệ thống, chúng sẽ kết hợp với nhau để tiêu thụ hết thời gian của CPU, và do đó các công việc chạy dài hạn sẽ *không bao giờ* nhận được bất kỳ thời gian nào của CPU (chúng bị “đói”). Ngay cả trong trường hợp này, ta sẽ muốn đạt được một số tiến bộ cho những công việc dài hạn này.

Thứ hai, một người dùng thông minh có thể viết lại chương trình của họ để lập lịch. Đánh lừa bộ lập lịch thường đề cập đến ý tưởng làm điều gì đó “lén lút” để lừa bộ lập lịch mang lại cho mình nhiều tài nguyên chia sẻ hơn là mức thông thường được nhận. Thuật toán ta đã mô tả dễ bị ảnh hưởng bời cuộc tấn công sau: trước khi lát cắt thời gian kết thúc, thực hiện thao tác I/O (đối với một số tệp không quan tâm), dó đó sẽ từ bỏ CPU. Làm như vậy cho phép ta ở trong cùng một hàng đợi và từ đó đạt được tỷ lệ phần trăm CPU cao hơn. Khi thực hiện đúng (VD: bằng cách chạy trong 99% lát cắt thời gian trước khi từ bỏ CPU), một công việc có thể gần như độc quyền CPU.

Cuối cùng, một chương trình có thể thay đổi hành vi của nó theo thời gian, những gì bị ràng buộc bởi CPU có thể chuyển sang giai đoạn tương tác. Với cách tiếp cận hiện tại, một công việc như vậy sẽ không may mắn và không được đối xử như những công việc tương tác khác trong hệ thống.

**Lịch trình phải được bảo mật khỏi cuộc “tấn công”**

Chúng ta có thể nghĩ rằng một chính sách lập lịch, bên trong chính hệ điều hành hoặc trong ngữ cảnh rộng hơn (VD: trong xử lý yêu cầu I/O của hệ thống lưu trữ phân tán), không phải là mối quan tâm về bảo mật. Nhưng trong nhiều trường hợp, nó chính xác là như vậy. Hãy xem xét đến trung tâm dữ liệu hiện đại, trong đó người dùng trên thế giới chia sẻ CPU, bộ nhớ, mạng và hệ thống lưu trữ; nếu không cẩn thận trong việc thiết kế và thục thi chính sách, một người dùng có thể gây bất lợi cho người khác và đạt được lợi ích cho chính người đó. Vì thế, chính sách lập lịch là một phần quan trọng của bảo mật của một hệ thống và cần được xây dựng cẩn thận.



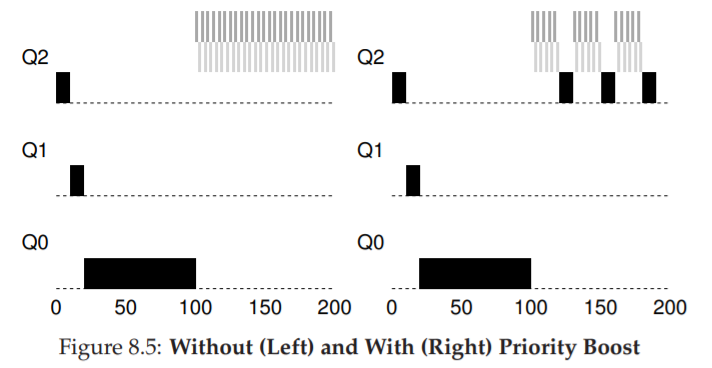
1. **Sự tăng cường ưu tiên**

Hãy thử thay đổi các quy tắc và xem liệu chúng ta có thể tránh được vấn đề “đói”. Ta có thể làm gì để đảm bảo rằng các công việc liên quan đến CPU sẽ đạt được một số tiến bộ. Ý tưởng đơn giản ở đây là định kỳ tăng cường mức độ ưu tiên của tất cả các công việc trong hệ thống. Có nhiều cách để đạt được điều này, nhưng ta hãy làm điều gì đó đơn giản: ném tất cả chúng vào hàng đợi trên cùng, từ đó, ta có một quy tắc mới:

* Quy tắc 5: Sau một khoảng thời gian S, di chuyển tất cả các công việc trong hệ thống đến hàng đợi trên cùng.

Quy tắc mới giải quyết hai vấn đề cùng một lúc. Đầu tiên, các quy trình được đảm bảo không bị đói: bằng cách ngồi ở hàng đợi trên cùng, một công việc sẽ chia sẻ CPU với các công việc có mức độ ưu tiên cao khác theo thuật toán Round Robin và từ đó, cuối cùng sẽ nhận được “phục vụ”. Thứ hai, nếu một công việc bị ràng buộc bởi tương tác CPU, bộ lập lịch xử lý nó đúng cách khi nó đã nhận được mức tăng ưu tiên.

Hãy xem một ví dụ. Trong trường hợp này, ta chỉ hiển thị hành vi của một công việc dài hạn khi cạnh tranh CPU với hai công việc tương tác ngắn hạn. Hai đồ thị được thể hiện trong hình 8.5. Bên trái, không có mức tăng ưu tiên và vì vậy, công việc ngắn hạn sẽ bị bỏ đói một lần khi hai công việc ngắn hạn đến. Ở bên phải, có một mức tăng ưu tiên sau mỗi 50ms (có thể là một giá trị quá nhỏ, nhưng sẽ sử dụng ở đây cho VD), và do đó, ít nhất đảm bảo rằng công việc dài hạn sẽ tạo ra một số tiến bộ, được nâng lên mức ưu tiên cao nhất sau mỗi 50ms và từ đó được chạy định kỳ.



Tất nhiên, việc thêm vào khoảng thời gian S dẫn đến một câu hỏi rõ ràng: S nên được đặt bao nhiêu? Nếu nó được đặt quá cao, công việc dài hạn có thể bị “đói”, quá thấp, các công việc mang tính tương tác có thể không nhận được sự chia sẻ của CPU.

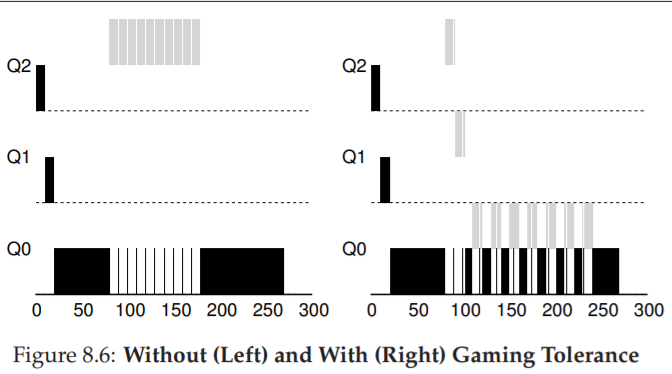
1. **Cải thiện việc giám sát hoạt động**

Hiện tại vẫn còn một vấn đề cần phải giải quyết: Làm thế nào để có thể ngăn chặn việc ăn gian bộ lập lịch. Vấn đề đó ở đây là quy tắc 4a và 4b, chính là để cho một công việc giữ lại thứ tự ưu tiên của nó bằng cách giải phóng CPU trước khi hết lát cát thời gian.

Phương án giải quyết ở đây là cải thiện giam sát thời gian CPU ở mỗi mức độ của MLFQ. Thay vì lẵng quên khoảng thời gian cần thiết để quy trình lát thời gian được sử dụng ở một mức độ đã cho, bộ lập lịch nên giám sát; mỗi khi một quy trình sử dụng khoảng phân bố của nó, nó được hạ cấp xuống hàng đợi ưu tên tiếp theo. Việc nó sử dụng liên tục các lát thời gian liên tục trong một khoảng thời gian dài hay ngắn không quan trọng. Vì thế, nên ta có thể viết lại quy tắc 4a và 4b thành quy tắc đơn sau đây:

* Quy tắc 4: Mỗi khi một công việc sử dụng hết khoảng thời gian phân bổ của nó ở một mức độ đã cho (kể cả nó có ngừng CPU bao nhiêu lần đi nữa), thì mức ưu tiên của nó đã bị hạ cấp (nói cách khác, nó bị hạ xuống một hàng đợi).

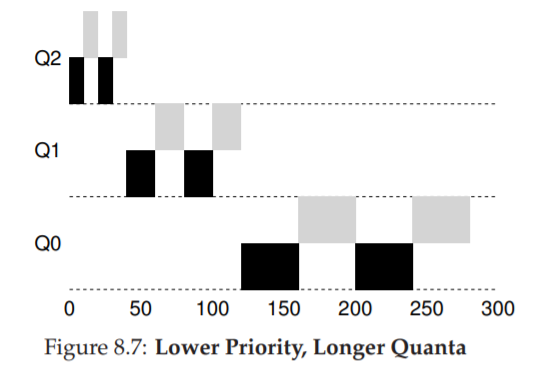
Hãy thử quan sát một ví dụ. Đồ thị 8.6 cho thấy những gì xảy ra khi một đơn vị công việc tìm cách ăn gian bộ lập lịch theo quy tắc 4a và 4b cũ (ở phía bên trái) cũng như quy tắc chống gian lận mới 4. Khi không có bất kỳ sự bảo vệ nào khỏi việc gian lận, một quy trình có thể phát ra một I/O ngay trước khi một lát thời gian kết thúc và sau đó chi phối thời gian CPU. Khi đã thiết lập bảo vệ , bất chấp hành vi I/O trong quy trình, hành vi I/O đó sẽ từ từ bị chuyển xuống hàng đợi sau, từ đó, không thể đạt được phần thời gian CPU nó không xứng đáng có được.



1. **Điều chỉnh MLFQ và những vấn đề khác**

Có một vài vấn đề phát sinh với việc lập lịch MLFQ. Một câu hỏi lớn đặt ra là làm cách nào để **thông số hóa** bộ lập lịch. Ví dụ, bao nhiêu nó nên chứa bao nhiêu hàng đợi? Mỗi hàng đợi nên có lát thời gian dài đến mức nào? Tần suất của việc gia tăng các mức ưu tiên sẽ phải như thế nào để có thể tránh sự cạn kiệt và giải thích cho những sự thay đổi hành vi? Không có câu trả lời dễ dàng cho những vấn đề này, và vì thế nên chỉ có một số trải nghiệm với đơn vị công việc và điều chỉnh tiếp nối mới có thể dẫn đến sự cân bằng đạt yêu cầu.

Ví dụ, hầu như tất cả các biến số MLFQ cho phép chiều dài lát thời gian biến đổi qua những hàng đợi khác nhau. Những hàng đợi có mức ưu tiên cao thường được cho những lát thời gian ngắn, chúng bao gồm những công việc mang tính tương tác cao, sau cùng thì, và vì thế nên có thể nhanh chóng biến đổi giữa việc chúng có ý nghĩa (ví dụ, 10 hoặc nhỏ hơn 10 mili giây). Ngược lại thì những hàng đợi có mức ưu tiên thấp chứa những công việc chạy lâu bị bó buộc bởi CPU; vì thế, thời gian lát thời gian hoạt động tốt càng được kéo dài (ví dụ, 100s của mili giây). Đồ thị 8.7 đã thể hiện một ví dụ khi mà 2 công việc được vận hành trong 20 mili giây ở hàng đợi cao nhất (với lát thời gian dài 10 mili giây), 40 mili giây ở giữa (lát thời gian dài 20 mili giây), và với lát thời gian dài 40 mili giây là thấp nhất.



Hệ thống xử lý Solaris MLFQ - lớp hệ lập lịch phân bổ thời gian -, hoặc TS - là tương đối dễ để cấu hình, nó cung cấp một hệ thống bảng có thể xác định chính các mức độ ưu tiên của một hệ thống được biến đổi qua suốt thời hạn của nó, độ dài của mỗi lát thời gian, và độ thường xuyên để tăng mức ưu tiên của một công việc; một quản trị viên có thể xử lí bảng này để làm hệ lập lệnh hoạt động theo nhiều cách khác nhau. Những giá trị mặc định cho bảng này là 60 hàng đợi, với chiều dài lát thời gian dần tăng từ 20 mili giây (mức ưu tiên cao nhất) tới vài trăm mili giây (thấp nhất), và mức ưu tiên được gia tăng trong khoảng mỗi một giây hay gì đó.

Những hệ lập lịch MLFQ khác không dùng một bảng hay những quy tắc chính xác được mô tả trong chương này; phần nào chúng điều chỉnh các mức độ ưu tiên bằng cách sử dụng những công thức toán học. Ví dụ, hệ lập lịch FreeBSD (bản 4.3) sử dụng một công thức để tính toán mức độ ưu tiên hiện tại của một công việc, căn cứ vào lượng CPU quy trình đó đã sử dụng; thêm vào đó, cách sử dụng bị mất dần theo thời gian, trong trường hợp ưu tiên mong đợi tăng theo một các khác so với những gì được mô tả trong tài liệu này. Xem tài liệu của Epema để thấy được một bản tổng quan xuất sắc của những thuật toán mất dần theo thời gian và các đặc tính của chúng.

Cuối cùng, nhiều bộ lập lịch có những tính năng khác mà bạn có thể gặp phải. Ví dụ, một số hệ lập lệnh đảo ngược những mức độ ưu tiên cao nhất cho hệ thống thực hiện làm việc; vì thế nên những người dùng điển hình không thể nào giành được mức độ ưu tiên cao nhất trong hệ thống. Một số hệ thống cũng cho người dùng lời khuyên để giúp họ thiết lập các ưu tiên; ví dụ, bằng cách sử dụng chức năng câu lệnh tốt, có thể tăng hoặc giảm mức độ ưu tiên của một công việc (đến mức độ nào đó) và từ đó tăng hoặc giảm cơ hội vận hành của nó ở bất kì thời gian nào đã cho.

1. **MLFQ: Tổng kết**

Ta đã mô tả một phương pháp tiếp cận lập lệnh được biết tới là Lập lịch với hàng đợi phản hồi đa cấp: MLFQ có nhiều mức độ của hàng đợi, và sử dụng phản hồi mức độ ưu tiên của một công việc đã cho, tập trung vào cách hoạt động của các công việc theo thời gian và xử lý chúng sao cho phù hợp.

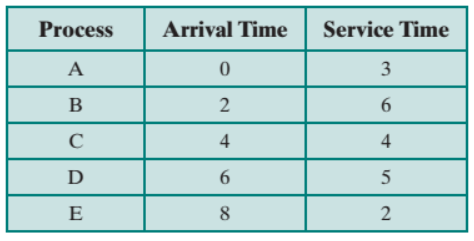
Các quy tắc của MLFQ:

* Quy tắc 1: Nếu mức ưu tiên(A) > mức ưu tiên(B), A chạy (B không chạy)
* Quy tắc 2: Nếu mức ưu tiên(A) = mức ưu tiên(B), A và B cùng chạy bằng giải thuật Roind Robin
* Quy tắc 3: Khi một công việc được đưa vào hệ thống, nó được đặt ở vị trí ưu tiên cao nhất (hàng đợi trên cùng).
* Quy tắc 4: Mỗi khi một công việc sử dụng hết khoảng thời gian phân bổ của nó ở một mức độ đã cho (kể cả nó có ngừng CPU bao nhiêu lần đi nữa), thì mức ưu tiên của nó đã bị hạ cấp (nói cách khác, nó bị hạ xuống một hàng đợi).
* Quy tắc 5: Sau một khoảng thời gian S, di chuyển tất cả các công việc trong hệ thống đến hàng đợi trên cùng.

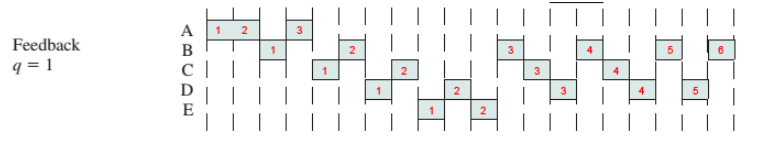
MLFQ thú vị vì lí do sau đây: thay vì yêu cầu khối kiến thức tiền nghiệm của bản chất một công việc, MLFQ quan sát sự thực hiện của một công việc và đặt mức ưu tiên của nó sao cho phù hợp. Bằng cách này, nó có thể đạt được những gì tốt nhất: có thể mang tới hiệu quả tổng quát xuất sắc (giống như SJF/STCF) cho những công việc ngắn hạn có độ tương tác cao, và công bằng cũng như tạo ra tiến bộ cho đơn vị công việc với CPU cường độ cao dài hạn. Chính vì lý do này, nhiều hệ thống, bao gồm phái sinh BSD UNIX Solaris và Windows NT và hệ thống thực hiện Windows theo sau sử dụng hình thức của MLFQ như hệ lập lịch nền.

# KẾT QUẢ CHẠY CHƯƠNG TRÌNH

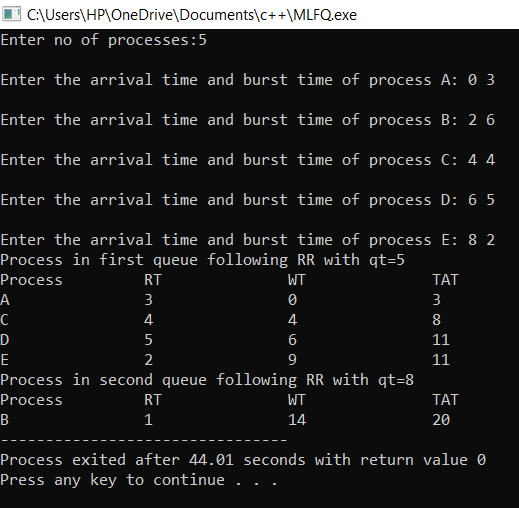
**Ví dụ:** Cho danh sách các tiến trình mà hệ điều hành phải lập lịch hoạt động với thông tin như sau:

****

**Sơ đồ thời gian:**

****

**Kết quả chạy chương trình:**

****

# PHẦN MỞ RỘNG:

# Lập lịch cho thời hạn sớm nhất trước tiên– Earliest Deadline First (EDF)

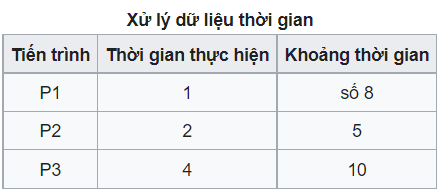
**Thời hạn sớm nhất trước tiên (EDF)** là một thuật toán lập lịch ưu tiên động tối ưu được sử dụng trong các hệ thống thời gian thực.  
Nó có thể được sử dụng cho cả lập lịch thời gian thực tĩnh và động.

EDF sử dụng mức độ ưu tiên cho các công việc để lập lịch trình. Nó chỉ định mức độ ưu tiên cho nhiệm vụ theo thời hạn tuyệt đối. Nhiệm vụ có thời hạn gần nhất sẽ được ưu tiên cao nhất. Các ưu tiên được phân công và thay đổi theo hướng năng động. EDF rất hiệu quả so với các thuật toán lập lịch khác trong hệ thống thời gian thực. Nó có thể làm cho việc sử dụng CPU lên khoảng 100% trong khi vẫn đảm bảo thời hạn của tất cả các tác vụ.

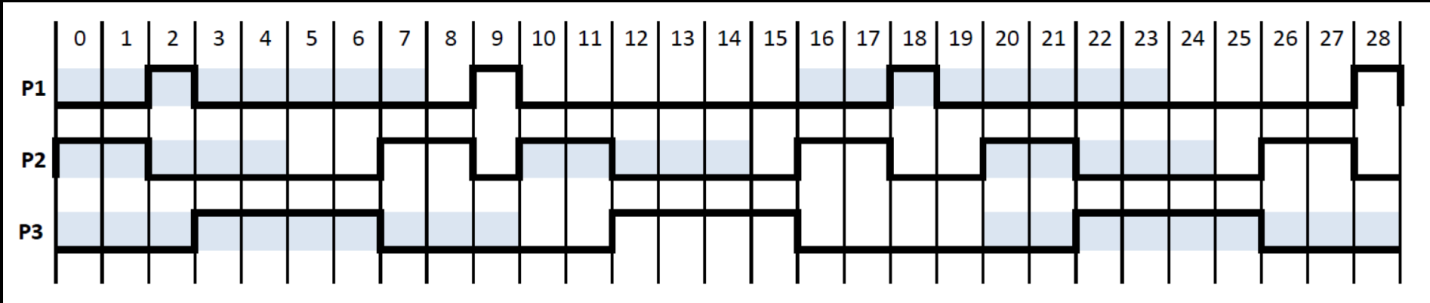
EDF bao gồm quá tải hạt nhân. Trong EDF, nếu mức sử dụng CPU nhỏ hơn 100%, thì điều đó có nghĩa là tất cả các tác vụ đã đáp ứng thời hạn. EDF tìm ra một lịch trình khả thi tối ưu. Lịch trình khả thi là lịch trình trong đó tất cả các tác vụ trong hệ thống được thực hiện trong thời hạn. Nếu EDF không thể tìm thấy một lịch trình khả thi cho tất cả các nhiệm vụ trong hệ thống thời gian thực, thì điều đó có nghĩa là không có thuật toán lập lịch tác vụ nào khác trong hệ thống thời gian thực có thể đưa ra một lịch trình khả thi. Tất cả các tác vụ đã sẵn sàng để thực thi phải thông báo thời hạn của chúng cho EDF khi tác vụ có thể chạy được.

Thuật toán lập lịch EDF không cần các nhiệm vụ hoặc quy trình phải định kỳ và các tác vụ hoặc quy trình cũng yêu cầu thời gian bùng nổ CPU cố định. Trong EDF, bất kỳ tác vụ đang thực thi nào cũng có thể được ưu tiên nếu bất kỳ phiên bản định kỳ nào khác có thời hạn sớm hơn đã sẵn sàng để thực thi và trở nên hoạt động. Ưu tiên được cho phép trong thuật toán lập lịch trình sớm nhất đến hạn chót đầu tiên.

Ví dụ: Hãy xem xét 3 quy trình định kỳ được lập lịch trên một bộ xử lý đơn ưu tiên. Thời gian và khoảng thời gian thực hiện được hiển thị trong bảng sau:



Trong ví dụ này, các đơn vị thời gian có thể được coi là [các lát thời gian](https://en.wikipedia.org/wiki/Time_slice) có thể lập lịch . Thời hạn là mỗi quy trình định kỳ phải hoàn thành trong khoảng thời gian của nó.

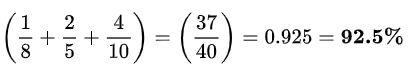
****

Trong biểu đồ thời gian trên, các cột đại diện cho các lát thời gian với thời gian tăng dần về bên phải và tất cả các quá trình đều bắt đầu khoảng thời gian của chúng ở lát thời gian 0. Màu xanh lam và trắng xen kẽ của biểu đồ thời gian cho biết từng khoảng thời gian của quá trình, với thời hạn thay đổi màu sắc.

Quy trình đầu tiên được lập lịch bởi EDF là P2, vì thời gian của nó là ngắn nhất và do đó nó có thời hạn sớm nhất. Tương tự như vậy, khi P2 hoàn thành, P1 được lên lịch, tiếp theo là P3.

Tại thời điểm 5, cả P2 và P3 đều có cùng thời hạn, cần phải hoàn thành trước thời điểm 10, vì vậy EDF có thể lên lịch cho một trong hai.

Việc sử dụng CPU sẽ là:

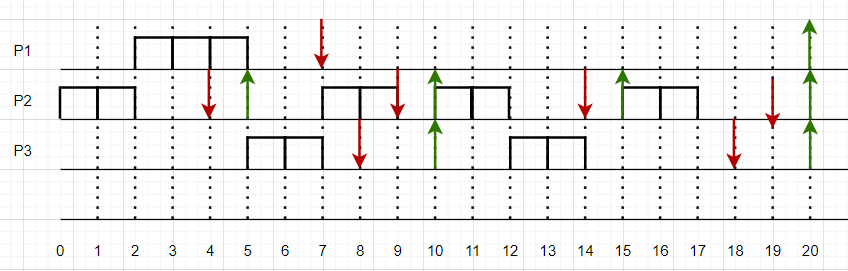
{\ displaystyle \ left ({\ frac {1} {8}} + {\ frac {2} {5}} + {\ frac {4} {10}} \ right) = \ left ({\ frac {37 } {40}} \ right) = 0.925 = {\ mathbf {92.5 \%}}}

Vì [bội số phổ biến nhất](https://en.wikipedia.org/wiki/Least_common_multiple) của các khoảng thời gian là 40, nên mẫu lập lịch có thể lặp lại sau mỗi 40 lát thời gian. Tuy nhiên, chỉ 37 trong số 40 lát thời gian đó được sử dụng bởi P1, P2 hoặc P3. Kể từ khi sử dụng, 92,5% và không lớn hơn 100%, hệ thống có thể lập lịch với EDF.

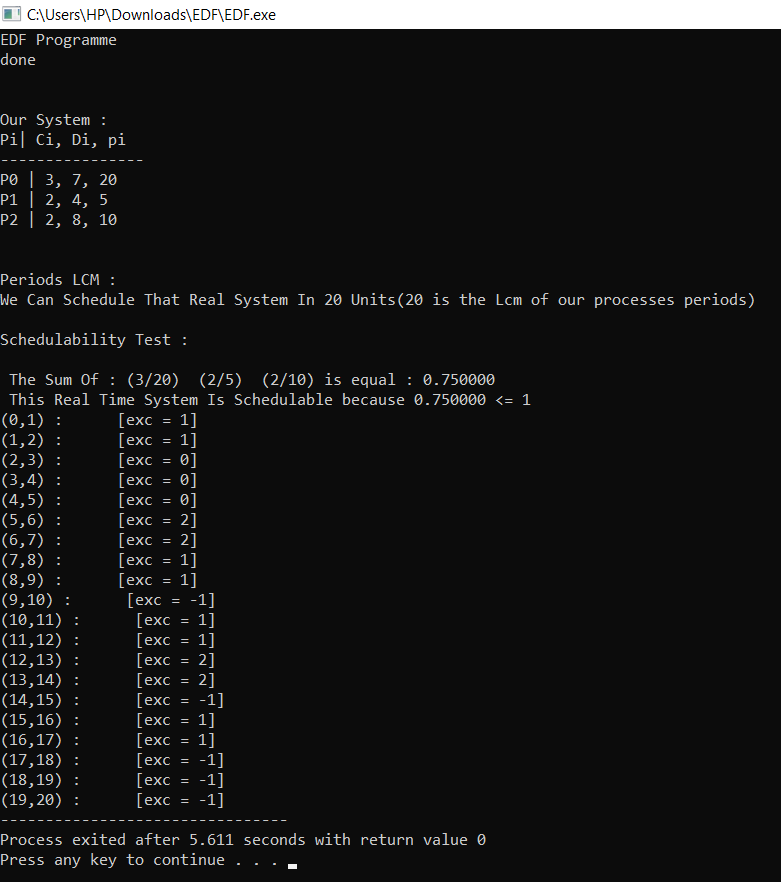
**Ví dụ:** Cho danh sách các tiến trình mà hệ điều hành phải lập lịch hoạt động với thông tin như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| **PROCESS** | **EX-TIME** | **DEADLINE** | **PERIOD** |
| **P1** | **3** | **7** | **20** |
| **P2** | **2** | **4** | **5** |
| **P3** | **2** | **8** | **10** |

**Sơ đồ thời gian:**



**Kết quả chạy chương trình:**

****