7

Lập lịch trình: Giới thiệu

Hiện tại, các cơ chế cấp thấp của các tiến trình đang chạy (ví dụ, chuyển đổi ngữ cảnh) phải rõ ràng; nếu không, hãy quay lại một hoặc hai chương và đọc lại mô tả về cách hoạt động của công cụ đó. Tuy nhiên, chúng tôi vẫn chưa hiểu các chính sách cấp cao mà bộ lập lịch hệ điều hành sử dụng. Bây giờ chúng tôi sẽ làm điều đó, trình bày một loạt các chính sách lập lịch trình (đôi khi được gọi là kỷ luật) mà nhiều người thông minh và chăm chỉ khác nhau đã áp dụng trong những năm qua.

Trên thực tế, nguồn gốc của lập kế hoạch có trước các hệ thống máy tính; các phương pháp tiếp cận ban đầu đã được thực hiện từ lĩnh vực quản lý hoạt động và áp dụng cho máy tính. Thực tế này không có gì đáng ngạc nhiên: các dây chuyền lắp ráp và nhiều nỗ lực khác của con người cũng đòi hỏi phải lên lịch trình và nhiều mối quan tâm tương tự tồn tại ở đó, bao gồm cả mong muốn hiệu quả giống như tia laser.
Và do đó, vấn đề của chúng tôi:

THE CRUX: CÁCH PHÁT TRIỂN CHÍNH SÁCH LỊCH TRÌNH

Làm thế nào chúng ta nên phát triển một khuôn khổ cơ bản để suy nghĩ về các chính sách lập lịch trình? Các giả định chính là gì? Số liệu nào là quan trọng? Những cách tiếp cận cơ bản nào đã được sử dụng trong hệ thống com puter sớm nhất?

7.1 Giả định về khối lượng công việc

Trước khi đi vào phạm vi các chính sách khả thi, trước tiên chúng ta hãy thực hiện một số giả định đơn giản hóa về các quy trình đang chạy trong hệ thống, đôi khi được gọi chung là khối lượng công việc. Xác định khối lượng công việc là một phần quan trọng trong việc xây dựng chính sách và bạn càng biết nhiều về khối lượng công việc, thì chính sách của bạn càng có thể được điều chỉnh tốt hơn.

Các giả định về khối lượng công việc mà chúng tôi đưa ra ở đây hầu hết là không thực tế, nhưng điều đó không sao cả (hiện tại), bởi vì chúng tôi sẽ thư giãn chúng khi chúng tôi tiếp tục, và thậm chí phát triển điều chỉnh những gì chúng tôi sẽ gọi là ... (tạm dừng kịch tính) ...

kỷ luật lập kế hoạch hoạt động đầy đủ1 .

Chúng tôi sẽ đưa ra các giả định sau về các quy trình, một số thời điểm được gọi là công việc, đang chạy trong hệ thống:

1. Mỗi công việc chạy trong một khoảng thời gian như nhau.
2. Tất cả các công việc đến cùng một lúc.
3. Sau khi bắt đầu, mỗi công việc sẽ hoàn thành.
4. Tất cả các công việc chỉ sử dụng CPU (tức là chúng không thực hiện I / O)
5. Thời gian thực hiện của mỗi công việc đã được biết trước.

Chúng tôi đã nói rằng nhiều giả định trong số này là không thực tế, nhưng cũng giống như một số động vật bình đẳng hơn những con khác trong Trang trại Động vật của Orwell [O45], một số giả định không thực tế hơn những giả định khác trong chương này. Cụ thể, bạn có thể bận tâm rằng thời gian thực hiện của mỗi công việc đã được biết trước: điều này sẽ làm cho bộ lập lịch trình trở nên toàn trí, mặc dù nó sẽ rất tuyệt vời (có thể xảy ra), nhưng không có khả năng xảy ra sớm.

7.2 Các chỉ số lập lịch

Ngoài việc đưa ra các giả định về khối lượng công việc, chúng tôi cũng cần một điều nữa để cho phép chúng tôi so sánh các chính sách lập lịch khác nhau: một quy trình lập kế hoạch đáp ứng. Số liệu chỉ là thứ mà chúng tôi sử dụng để đo lường điều gì đó và có một số số liệu khác nhau có ý nghĩa trong việc lập lịch trình.

Tuy nhiên, hiện tại, chúng ta hãy đơn giản hóa cuộc sống của mình bằng cách đơn giản chỉ có một thước đo tội lỗi: thời gian quay vòng. Thời gian quay vòng của một công việc được định nghĩa là thời gian mà công việc đó hoàn thành trừ đi thời gian mà công việc đó đến trong hệ thống. Chính thức hơn, thời gian quay vòng Tturnaround là:

$$T_{turnaround} = T_{hoàn thành} - T_{arrival} \tag{7,1}$$

Bởi vì chúng tôi đã giả định rằng tất cả các công việc đến cùng một lúc, hiện tại Tarrival = 0 và do đó Tturnaround = Tcompletion. Thực tế này sẽ thay đổi khi chúng ta nới lỏng các giả định nói trên.

Bạn nên lưu ý rằng thời gian quay vòng là một số liệu hiệu suất , sẽ là trọng tâm chính của chúng tôi trong chương này. Một số liệu quan tâm khác là công bằng, như được đo lường (ví dụ) bằng Chỉ số Công bằng của Jain [J91]. Perfor mance và công bằng thường mâu thuẫn trong việc lập lịch trình; một công cụ lập lịch biểu, đối với nhiều công việc, có thể tối ưu hóa hiệu suất nhưng với cái giá phải trả là ngăn một số công việc chạy, do đó làm giảm tính công bằng. Câu hỏi hóc búa này cho chúng ta thấy rằng cuộc sống không phải lúc nào cũng hoàn hảo.

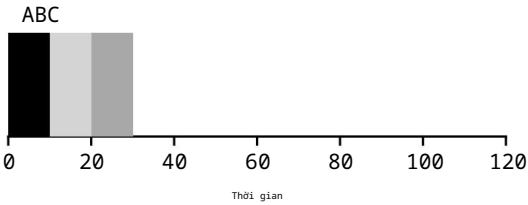
7.3 Xuất trước, xuất trước (FIFO)

Thuật toán cơ bản nhất mà chúng ta có thể triển khai được gọi là lập lịch trình First In, First Out (FIFO) hoặc đôi khi là First Come, First Served (FCFS).

[1] Nói theo cách giống như bạn sẽ nói "Một Ngôi sao Tử thần hoạt động hoàn chỉnh."

FIFO có một số đặc tính tích cực: nó rõ ràng là đơn giản và do đó
dễ để thực hiện. Và, với những giả định của chúng tôi, nó hoạt động khá tốt.

Chúng ta hãy làm một ví dụ nhanh với nhau. Hãy tưởng tượng ba công việc đến
hệ thống, A, B và C, gần như cùng một lúc (Tarrival = 0). Tại vì
FIFO phải đặt một số công việc lên hàng đầu, hãy giả sử rằng trong khi tất cả họ đều đến
đồng thời, A đến chỉ một sợi tóc trước khi B đến chỉ một sợi tóc
trước C. Cũng giả sử rằng mỗi công việc chạy trong 10 giây. Cái gì sẽ
thời gian quay vòng trung bình cho những công việc này?



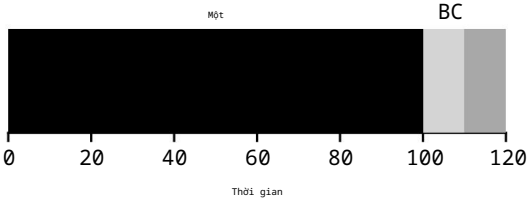
Hình 7.1: Ví dụ đơn giản về FIFO

Từ Hình 7.1, bạn có thể thấy rằng A đã hoàn thành ở mức 10, B ở mức 20 và C ở mức 30.
Do đó, thời gian quay vòng trung bình cho ba công việc chỉ đơn giản là 20. $\frac{10 + 20 + 30}{3} =$
Tính toán thời gian quay vòng dễ dàng như vậy.

Bây giờ chúng ta hãy thư giãn một trong những giả định của chúng ta. Đặc biệt, chúng ta hãy
thư giãn với tư cách là nhiệm vụ 1, và do đó không còn giả định rằng mỗi công việc chạy cho cùng một
khoảng thời gian. FIFO hoạt động như thế nào bây giờ? Loại khối lượng công việc
bạn có thể xây dựng để làm cho FIFO hoạt động kém?

(hãy nghĩ về điều này trước khi đọc tiếp ... tiếp tục suy nghĩ ... hiểu chưa ?!)

Có lẽ bạn đã hiểu ra điều này ngay bây giờ, nhưng để đề phòng, chúng ta hãy làm
một ví dụ cho thấy các công việc có độ dài khác nhau có thể dẫn đến rắc rối như thế nào đối với
Lập lịch FIFO. Cụ thể, chúng ta hãy giả sử ba công việc (A, B và
C), nhưng lần này A chạy trong 100 giây trong khi B và C chạy trong 10 giây mỗi người.



Hình 7.2: Tại sao FIFO không tuyệt vời như vậy

Như bạn có thể thấy trong Hình 7.2, Công việc A chạy đầu tiên trong 100 giây đầy đủ
trước khi B hoặc C thậm chí có cơ hội để chạy. Do đó, vòng quay trung bình
thời gian cho hệ thống cao: 110 giây đau đớn (Vấn đề này $\frac{100 + 110 + 120}{3} = 110$).

thường được gọi là hiệu ứng đoàn xe [B + 79], trong đó
một số người tiêu dùng tiềm năng tương đối ngắn của tài nguyên được xếp hàng đợi

MẸO: NGUYÊN TẮC CỦA SJF

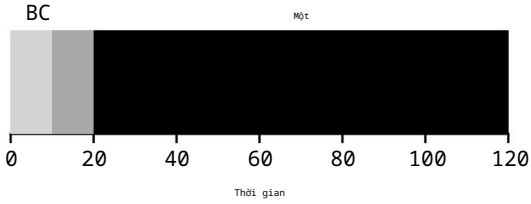
Công việc ngắn nhất Đầu tiên đại diện cho một nguyên tắc lập kế hoạch chung có thể được áp dụng cho bất kỳ hệ thống nào mà thời gian quay vòng nhận thức được trên mỗi khách hàng (hoặc, trong trường hợp của chúng tôi, một công việc) quan trọng. Hãy nghĩ về bất kỳ dòng nào bạn đã chờ đợi: nếu cơ sở được đề cập quan tâm đến sự hài lòng của khách hàng, có thể họ đã tính đến SJF. Ví dụ: các cửa hàng tạp hóa thường có dòng "mười mặt hàng trở xuống" để đảm bảo rằng những người mua sắm chỉ với một số ít những thứ cần mua không bị mắc kẹt sau khi gia đình chuẩn bị cho một số mùa đông hạt nhân sắp tới.

đằng sau một người tiêu dùng tài nguyên nặng ký. Kịch bản lập lịch trình này có thể nhắc nhở bạn về một hàng duy nhất tại một cửa hàng tạp hóa và bạn cảm thấy như thế nào khi bạn nhìn thấy người trước mặt bạn với ba chiếc xe đẩy đầy hàng và một sổ séc ra; nó sẽ mất một thời gian[2] .

Vậy chúng ta nên làm gì? Làm cách nào chúng tôi có thể phát triển một thuật toán tốt hơn để đối phó với thực tế mới của chúng ta về các công việc chạy trong những khoảng thời gian khác nhau? Hãy nghĩ về nó trước; sau đó đọc tiếp.

7.4 Công việc ngắn nhất đầu tiên (SJF)

Nó chỉ ra rằng một cách tiếp cận rất đơn giản giải quyết vấn đề này; trong thực tế nó là một ý tưởng bị đánh cắp từ nghiên cứu hoạt động [C54, PV56] và được áp dụng cho lập lịch các công việc trong hệ thống máy tính. Kỷ luật lập lịch mới này được gọi là Công việc ngắn nhất Đầu tiên (SJF) và tên này phải dễ hiểu hãy nhớ vì nó mô tả chính sách khá đầy đủ: nó chạy công việc ngắn nhất đầu tiên, sau đó là công việc ngắn nhất tiếp theo, v.v.



Hình 7.3: Ví dụ đơn giản về SJF

Hãy lấy ví dụ của chúng tôi ở trên nhưng với SJF là chính sách lên lịch của chúng tôi. Hình 7.3 cho thấy kết quả của việc chạy A, B và C. Hy vọng rằng dia gram cho thấy rõ lý do tại sao SJF hoạt động tốt hơn nhiều liên quan đến thời gian quay vòng theo độ tuổi. Đơn giản bằng cách chạy B và C trước A, SJF giảm thời gian quay vòng trung bình từ 110 giây đến 50 ($\frac{10 + 20 + 120}{3}$ = 50), nhiều hơn một yếu tố của hai cải tiến.

2 Hành động được đề xuất trong trường hợp này: nhanh chóng chuyển sang một dòng khác hoặc mất nhiều thời gian, thở sâu và thư giãn. Đúng vậy, hít vào, thở ra. Sẽ ổn thôi, đừng lo lắng.

BÊN TRONG: LỊCH TRÌNH TRƯỚC

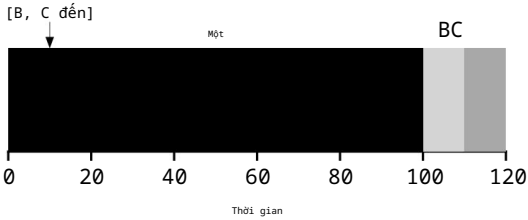
Trong những ngày cũ của điện toán hàng loạt, một số bộ điều khiển lịch trình không đặt trước đã được phát triển; những hệ thống như vậy sẽ chạy từng công việc để hoàn thành trước khi cân nhắc có nên chạy một công việc mới hay không. Hầu như tất cả các trình quản lý lịch hiện đại đều được ưu tiên và khá sẵn sàng dừng một quá trình chạy ning để chạy một quá trình khác. Điều này ngụ ý rằng bộ lập lịch sử dụng các cơ chế mà chúng tôi đã tìm hiểu trước đây; đặc biệt, bộ lập lịch có thể thực hiện chuyển đổi ngữ cảnh, tạm thời dừng một quá trình đang chạy và tiếp tục (hoặc bắt đầu) khác.

Trên thực tế, với giả định của chúng tôi về các công việc đều đến cùng một lúc, chúng tôi có thể chứng minh rằng SJF thực sự là một thuật toán lập lịch tối ưu . Đã bao giờ, bạn đang ở trong một lớp học hệ thống, không phải nghiên cứu lý thuyết hoặc hoạt động; không được phép chứng minh.

Vì vậy, chúng tôi đi đến một cách tiếp cận tốt để lên lịch với SJF, nhưng các giả định vẫn còn khá phi thực tế. Hãy thư giãn khác. Đặc biệt, chúng ta có thể nhắm mục tiêu giả định 2 và bây giờ giả định rằng công việc có thể đến ở bất kỳ thời gian thay vì tất cả cùng một lúc. Điều này dẫn đến những vấn đề gì?

(Một lần nữa dừng lại để suy nghĩ ... bạn đang nghĩ sao? Nào, bạn có thể làm được)

Ở đây chúng ta có thể minh họa vấn đề một lần nữa bằng một ví dụ. Thời gian này, giả sử A đến t = 0 và cần chạy trong 100 giây, trong khi B và C đến lúc t = 10 và mỗi xe cần chạy trong 10 giây. Với tinh khiết SJF, chúng ta sẽ xem lịch trình trong Hình 7.4.



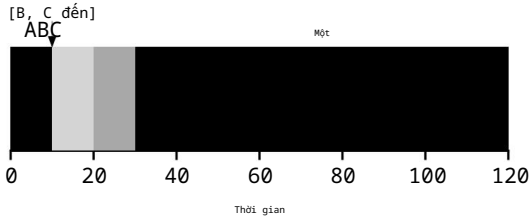
Hình 7.4: SJF với các chuyến đến muộn từ B và C

Như bạn có thể thấy trong hình, mặc dù B và C đến trong thời gian ngắn sau khi A, họ vẫn bị buộc phải đợi cho đến khi A hoàn thành, và do đó phải chịu cùng một vấn đề đoàn xe. Thời gian quay vòng trung bình cho ba công việc này là 103,33 giây (). Người lập lịch có thể làm gì?

$$\frac{100 + (110 - 10) + (120 - 10)}{3}$$

7.5 Thời gian hoàn thành đầu tiên ngắn nhất (STCF)

Để giải quyết mối quan tâm này, chúng ta cần nới lỏng giả định 3 (rằng các công việc phải chạy đến khi hoàn thành), vì vậy hãy làm điều đó. Chúng tôi cũng cần một số máy móc bên trong chính trình lập lịch trình. Như bạn có thể đã đoán, với những thông tin trước đây của chúng tôi về ngắt bộ hẹn giờ và chuyển đổi ngữ cảnh, bộ lập lịch có thể



Hình 7.5: Ví dụ đơn giản về STCF

chắc chắn làm điều gì đó khác khi B và C đến: nó có thể báo trước công việc A và quyết định chạy một công việc khác, có lẽ sẽ tiếp tục A sau đó. SJF theo quan điểm của chúng tôi là một bộ lập lịch không ưu tiên , và do đó gặp phải các vấn đề miêu tả trên.

May mắn thay, có một bộ lập lịch thực hiện chính xác điều đó: thêm preemp tion vào SJF, được gọi là Thời gian hoàn thành đầu tiên ngắn nhất (STCF) hoặc Bộ lập lịch công việc đầu tiên (PSJF) dự phòng ngắn nhất [CK68]. Bất cứ lúc nào một cái mới công việc vào hệ thống, bộ lập lịch STCF xác định công việc nào trong số các công việc điều chỉnh lại (bao gồm cả công việc mới) còn lại ít thời gian nhất và lên lịch cái đó. Do đó, trong ví dụ của chúng tôi, STCF sẽ chặn A và chạy B và C để hoàn thành; chỉ khi chúng kết thúc thì thời gian còn lại của A mới là lên kế hoạch. Hình 7.5 cho thấy một ví dụ.

Kết quả là thời gian quay vòng trung bình được cải thiện nhiều: 50 giây ($\frac{(120\ 0) + (20\ 10) + (30\ 10)}{3}$). Và như trước đây, với những giả định mới của chúng tôi, STCF là tối ưu có thể chứng minh được; cho rằng SJF là tối ưu nếu tất cả các công việc đến đồng thời, bạn có thể có thể nhìn thấy trực giác đằng sau tính tối ưu của STCF.

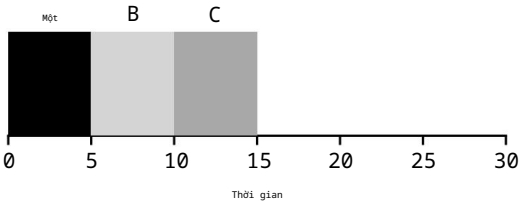
7.6 Một số liệu mới: Thời gian phản hồi

Do đó, nếu chúng ta biết thời lượng công việc và công việc đó chỉ sử dụng CPU, và chỉ số liệu là thời gian quay vòng, STCF sẽ là một chính sách tuyệt vời. Trong thực tế, đối với một số hệ thống tính toán hàng loạt ban đầu, các loại lập lịch này các thuật toán có ý nghĩa. Tuy nhiên, sự ra đời của chia sẻ thời gian máy móc đã thay đổi tất cả những điều đó. Giờ đây, người dùng sẽ ngồi tại một thiết bị đầu cuối và bắt buộc hiệu suất tương tác từ hệ thống. Và do đó, một số liệu được sinh ra: thời gian phản hồi.

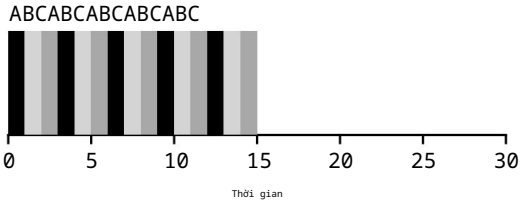
Chúng tôi định nghĩa thời gian phản hồi là thời gian kể từ khi công việc đến hệ thống lần đầu tiên được lên lịch[3] . Chính thức hơn:

$$T_{response} = T_{firstrun} - T_{arrival} \quad (7.2)$$

[3] Một số định nghĩa nó hơi khác một chút, ví dụ, bao gồm cả thời gian cho đến khi công việc tạo ra một số loại "phản ứng"; định nghĩa của chúng tôi là phiên bản trường hợp tốt nhất của điều này, về cơ bản giả định rằng công việc tạo ra phản hồi ngay lập tức.



Hình 7.6: Một lần nữa SJF (Kém thời gian đáp ứng)



Hình 7.7: Vòng quay vòng (Tốt cho thời gian phản hồi)

Ví dụ, nếu chúng ta có lịch trình từ Hình 7.5 (với A đến
tại thời điểm 0, và B và C tại thời điểm 10), thời gian phản hồi của mỗi công việc là
sau: 0 cho công việc A, 0 cho B và 10 cho C (trung bình: 3,33).

Như bạn có thể nghĩ, STCF và các ngành liên quan không tốt về thời gian phản hồi.
Nếu ba công việc đến cùng một lúc,
ví dụ: công việc thứ ba phải đợi hai công việc trước chạy trong
toàn bộ của chúng trước khi được lên lịch chỉ một lần. Mặc dù tuyệt vời để quay vòng
thời gian, cách tiếp cận này khá tệ đối với thời gian phản hồi và tính tương tác. Trong
hành động, hãy tưởng tượng bạn đang ngồi ở một thiết bị đầu cuối, đánh máy và phải đợi 10 giây
để xem phản hồi từ hệ thống chỉ vì một số công việc khác đã lên lịch trước mắt bạn:
không quá dễ chịu.

Vì vậy, chúng tôi còn lại với một vấn đề khác: làm thế nào chúng tôi có thể xây dựng một bộ lập lịch
nhạy cảm với thời gian phản hồi?

7.7 Vòng quay

Để giải quyết vấn đề này, chúng tôi sẽ giới thiệu một thuật toán lập lịch mới,
thường được gọi là lập lịch Round-Robin (RR) [K64]. Cơ bản
ý tưởng rất đơn giản: thay vì chạy công việc để hoàn thành, RR chạy một công việc cho một
lát thời gian (đôi khi được gọi là lượng tử lập lịch) và sau đó chuyển
đến công việc tiếp theo trong hàng đợi chạy. Nó lặp đi lặp lại như vậy cho đến khi công việc được
hoàn thành. Vì lý do này, RR đôi khi được gọi là cắt thời gian. Lưu ý rằng
độ dài của một lát thời gian phải là bội số của khoảng thời gian ngắt của bộ định thời;
do đó, nếu bộ hẹn giờ ngắt sau mỗi 10 mili giây, thì phần thời gian có thể là
10, 20 hoặc bất kỳ bội số nào khác của 10 ms.

Để hiểu RR chi tiết hơn, chúng ta hãy xem một ví dụ. Giả định
ba công việc A, B và C đến cùng một lúc trong hệ thống và

MẸO: HẠN CHẾ CÓ THỂ GIẢM CHI PHÍ

Kỹ thuật khấu hao chung thường được sử dụng trong các hệ thống
khi có một chi phí cố định cho một số hoạt động. Bằng cách gánh chịu chi phí đó ít hơn
thường (tức là, bằng cách thực hiện thao tác ít lần hơn), tổng chi phí cho
hệ thống bị giảm. Ví dụ: nếu lát thời gian được đặt thành 10 mili giây và
Chi phí chuyển đổi ngữ cảnh là 1 mili giây, khoảng 10% thời gian được sử dụng để chuyển đổi ngữ
cảnh và do đó bị lãng phí. Nếu chúng tôi muốn khấu hao chi phí này, chúng tôi có thể tăng
lát cắt thời gian, ví dụ, đến 100 mili giây. Trong trường hợp này, ít hơn 1% thời gian được sử dụng
chuyển đổi ngữ cảnh, và do đó chi phí cắt thời gian đã được khấu hao.

mỗi người muốn chạy trong 5 giây. Một bộ lập lịch SJF chạy từng công việc để
hoàn thành trước khi chạy khác (Hình 7.6). Ngược lại, RR với
khoảng thời gian 1 giây sẽ chuyển qua các công việc một cách nhanh chóng (Hình 7.7).

Thời gian phản hồi trung bình của RR là: $\frac{0 + 1 + 2}{3} = 1$; đối với SJF, trung bình lại thời gian tài trợ là: $\frac{0 + 5 + 10}{3} = 5$.

Như bạn có thể thấy, độ dài của lát thời gian rất quan trọng đối với RR. Ngắn hơn
chính là, hiệu suất của RR theo chỉ số thời gian phản hồi càng tốt.
Tuy nhiên, làm cho thời gian quá ngắn là một vấn đề: đột nhiên
chi phí chuyển đổi ngữ cảnh sẽ chi phối hiệu suất tổng thể. Do đó, việc căn cứ vào độ dài của
khoảng thời gian thể hiện một sự đánh đổi đối với người ký hệ thống, làm cho nó đủ dài để phân bổ
chi phí chuyển đổi mà không cần
làm cho nó quá lâu khiến hệ thống không còn phản hồi.

Lưu ý rằng chi phí chuyển đổi ngữ cảnh không chỉ phát sinh từ
Hệ điều hành hành động lưu và khôi phục một số đăng ký. Khi các chương trình chạy,
chúng tạo ra rất nhiều trạng thái trong bộ nhớ đệm CPU, TLB, bộ dự đoán nhánh,
và phần cứng trên chip khác. Chuyển sang công việc khác gây ra trạng thái này
được xóa và trạng thái mới có liên quan đến công việc hiện đang chạy
mang lại, có thể chính xác là một chi phí hiệu suất đáng chú ý [MB91].

RR, với một khoảng thời gian hợp lý, do đó, là một công cụ lập lịch xuất sắc nếu thời gian tài
trợ lại là số liệu duy nhất của chúng tôi. Nhưng còn người bạn cũ của chúng ta thì sao
thời gian? Hãy xem lại ví dụ của chúng tôi ở trên. A, B và C, mỗi thứ có thời gian chạy là 5 giây,
đến cùng một lúc và RR là bộ lập lịch
với lát thời gian (dài) 1 giây. Chúng ta có thể thấy từ hình trên rằng
A kết thúc ở vị trí 13, B ở vị trí 14 và C ở vị trí 15, với mức trung bình là 14. Khá khủng khiếp!

Do đó, không có gì ngạc nhiên khi RR thực sự là một trong những chính sách tồi tệ nhất nếu
thời gian quay vòng là số liệu của chúng tôi. Về mặt trực giác, điều này sẽ có ý nghĩa: cái gì
RR đang làm là kéo dài từng công việc miễn là nó có thể, bằng cách chỉ chạy
mỗi công việc trong một thời gian ngắn trước khi chuyển sang công việc tiếp theo. Bởi vì quay vòng
thời gian chỉ quan tâm đến khi công việc kết thúc, RR gần như bằng số thập phân, thậm chí còn tệ hơn
hơn FIFO đơn giản trong nhiều trường hợp.

Nói chung hơn, bất kỳ chính sách nào (chẳng hạn như RR) công bằng, tức là phân chia đều CPU
giữa các quy trình đang hoạt động trên một quy mô thời gian nhỏ, sẽ thực hiện
kém về các chỉ số, chẳng hạn như thời gian quay vòng. Thật vậy, đây là một
đánh đổi: nếu bạn sẵn sàng không công bằng, bạn có thể chạy những công việc ngắn hơn để thỏa mãn,
nhưng với chi phí là thời gian phản hồi; nếu thay vào đó bạn coi trọng sự công bằng,

MẸO: TIỆN ÍCH CAO HƠN TIỆN ÍCH KHI CÓ THỂ , hãy
chồng chéo các hoạt động để tối đa hóa việc sử dụng các tem hệ thống. Chồng chéo hữu ích trong nhiều lĩnh vực khác nhau, bao gồm cả khi tạo I / O đĩa hoặc gửi tin nhắn đến các máy từ xa; trong cả hai trường hợp, bắt đầu hoạt động và sau đó chuyển sang công việc khác là một ý tưởng hay, và cải thiện việc sử dụng và hiệu quả tổng thể của hệ thống.

thời gian phản hồi được giảm xuống, nhưng với chi phí của thời gian quay vòng. Kiểu đánh đổi này là phổ biến trong các hệ thống; bạn không thể có bánh của bạn và ăn nó quá 4 .

Chúng tôi đã phát triển hai loại lịch biểu. Loại đầu tiên (SJF, STCF) tối ưu hóa thời gian quay vòng, nhưng không tốt cho thời gian phản hồi. Loại thứ hai (RR) tối ưu hóa thời gian phản hồi nhưng không tốt cho việc quay vòng. Và chúng ta vẫn có hai giả định cần được nới lỏng: giả định 4 (rằng các công việc không có I / O) và giả định 5 (rằng thời gian thực hiện của mỗi công việc đã được biết trước).
Hãy giải quyết những giả định đó tiếp theo.

7.8 Kết hợp I / O

Đầu tiên, chúng ta sẽ thả lỏng giả định 4 - tất nhiên tất cả các chương trình đều thực hiện I / O. Hãy tưởng tượng một chương trình không nhận bất kỳ đầu vào nào: nó sẽ tạo ra cùng một đầu ra mỗi lần. Hãy tưởng tượng một thứ không có đầu ra: đó là cây tục ngữ rơi trong rừng, không ai nhìn thấy; không quan trọng là nó đã chạy.

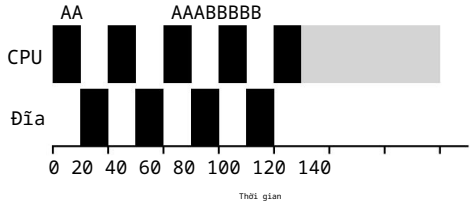
Người lập lịch rõ ràng có quyết định đưa ra khi một công việc bắt đầu một yêu cầu I / O, bởi vì công việc hiện đang chạy sẽ không sử dụng CPU khi nhập I / O; nó bị chặn khi chờ I / O hoàn thành. Nếu I / O được gửi đến ổ đĩa cứng, quá trình có thể bị chặn trong vài mili giây hoặc lâu hơn, tùy thuộc vào tải I / O hiện tại của ổ đĩa. Do đó, trình uler lập lịch có thể sẽ lên lịch cho một công việc khác trên CPU tại thời điểm đó.

Người lập lịch cũng phải đưa ra quyết định khi I / O hoàn tất.
Khi điều đó xảy ra, một ngắt sẽ xuất hiện và HĐH sẽ chạy và chuyển quá trình đã cấp I / O từ bị chặn trở lại trạng thái sẵn sàng. Tất nhiên, nó thậm chí có thể quyết định vận hành công việc tại thời điểm đó. Hệ điều hành nên xử lý từng công việc như thế nào?

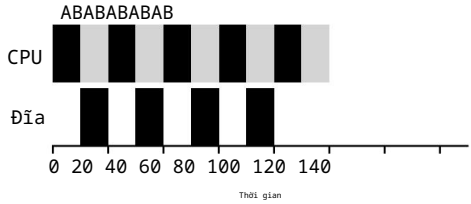
Để hiểu rõ hơn vấn đề này, chúng ta hãy giả sử chúng ta có hai công việc, A và B, mỗi công việc cần 50 ms thời gian CPU. Tuy nhiên, có một sự khác biệt rõ ràng: A chạy trong 10 ms và sau đó đưa ra yêu cầu I / O (giả sử ở đây rằng mỗi I / Os mất 10 ms), trong khi B chỉ sử dụng CPU trong 50 ms và không thực hiện I / O. Bộ lập lịch chạy A trước, B sau (Hình 7.8).

Giả sử chúng tôi đang cố gắng xây dựng bộ lập lịch STCF. Trình lập lịch biểu như vậy nên tính toán như thế nào đối với thực tế là A được chia thành 5 công việc phụ 10 mili giây,

4 Một câu nói khiến mọi người bối rối, vì lẽ ra phải là "Bạn không thể giữ chiếc bánh của mình và ăn nó nữa" (đó là điều hiển nhiên phải không?). Thật ngạc nhiên, có một trang wikipedia nói về câu nói này; tuyệt vời hơn nữa, thật là thú vị khi đọc [W15]. Như họ nói bằng tiếng Ý, bạn không thể Avere la botte piena e la moglie ubriaca.



Hình 7.8: Sử dụng tài nguyên kém



Hình 7.9: Chồng chéo cho phép sử dụng tài nguyên tốt hơn

trong khi B chỉ là một nhu cầu CPU 50 ms duy nhất? Rõ ràng, việc chỉ chạy một công việc rồi đến công việc khác mà không tính đến cách tính đến I / O chẳng có ý nghĩa gì.

Một cách tiếp cận phổ biến là coi mỗi công việc phụ 10 ms của A như một công việc sâu sắc. Do đó, khi hệ thống khởi động, lựa chọn của nó là lên lịch 10 ms A hay 50 ms B. Với STCF, sự lựa chọn rất rõ ràng: chọn cái ngắn hơn, trong trường hợp này là A. Sau đó, khi con đầu tiên công việc của A đã hoàn thành, chỉ còn lại B và nó bắt đầu chạy. Sau đó, một công việc phụ mới của A được gửi, và nó sẽ vượt qua B và chạy trong 10 mili giây. Làm như vậy cho phép chồng chéo, với CPU được sử dụng bởi một quy trình trong khi chờ I / O của một quy trình khác hoàn thành; do đó hệ thống được sử dụng tốt hơn (xem Hình 7.9).

Và do đó, chúng tôi thấy cách một bộ lập lịch có thể kết hợp I / O. Bằng cách coi mỗi vụ nổ CPU là một công việc, bộ lập lịch đảm bảo các quy trình “hoạt động kém hiệu quả” được chạy thường xuyên. Trong khi các công việc tương tác đó đang thực hiện I / O, các công việc khác sử dụng nhiều CPU sẽ chạy, do đó sử dụng bộ xử lý tốt hơn.

7.9 Không còn Oracle

Với một cách tiếp cận cơ bản cho I / O, chúng tôi đi đến giả định cuối cùng của chúng tôi: rằng người lập lịch biết thời lượng của mỗi công việc. Như chúng tôi đã nói trước đây, đây có thể là giả định tồi tệ nhất mà chúng tôi có thể đưa ra. Trên thực tế, trong một HĐH mục đích chung (như những HĐH chúng ta quan tâm), HĐH thường biết rất ít về độ dài của mỗi công việc. Vì vậy, làm thế nào chúng ta có thể xây dựng một cách tiếp cận giống như SJF / STCF mà không có kiến thức tiên nghiệm như vậy? Hơn nữa, làm thế nào chúng ta có thể kết hợp một số ý tưởng mà chúng ta đã thấy với bộ lập lịch RR để thời gian phản hồi cũng khá tốt?

7.10 Tóm tắt

Chúng tôi đã giới thiệu những ý tưởng cơ bản đằng sau việc lập lịch và phát triển hai nhóm phương pháp tiếp cận. Đầu tiên chạy công việc ngắn nhất còn lại và do đó tối ưu hóa thời gian quay vòng; thứ hai xen kẽ giữa tất cả các công việc và do đó tối ưu hóa thời gian phản hồi. Cả hai đều xấu trong khi cái kia tốt, than ôi, một sự đánh đổi cố hữu phổ biến trong các hệ thống. Chúng tôi cũng đã thấy cách chúng tôi có thể kết hợp I / O vào bức tranh, nhưng vẫn chưa giải quyết được vấn đề về khả năng cơ bản của hệ điều hành không thể nhìn thấy trong tương lai. Trong thời gian ngắn, chúng ta sẽ xem cách khắc phục vấn đề này, bằng cách xây dựng một công cụ lập lịch sử dụng quá khứ gần đây để dự đoán tương lai. Bộ lập lịch này được gọi là hàng đợi phản hồi đa cấp và nó là chủ đề của chương tiếp theo.

Người giới thiệu

[B + 79] “Hiện tượng đoàn xe” của M. Blasgen, J. Gray, M. Mitoma, T. Price. ACM Op erating Systems Review, 13: 2, tháng 4 năm 1979. Có lẽ là tài liệu tham khảo đầu tiên về các đoàn xe, xuất hiện trong cơ sở dữ liệu cũng như hệ điều hành.

[C54] “Chỉ định ưu tiên trong các vấn đề về hàng đợi” của A. Cobham. Tạp chí Nghiên cứu Hoạt động, 2:70, trang 70–76, 1954. Bài báo tiên phong về việc sử dụng phương pháp tiếp cận SJF trong việc lên lịch sửa chữa máy móc.

[K64] “Phân tích bộ xử lý chia sẻ thời gian” của Leonard Kleinrock. Naval Research Logistics Quarterly, 11: 1, trang 59–73, tháng 3 năm 1964. Có thể là tài liệu tham khảo đầu tiên về thuật toán lập lịch trình vòng tròn; chắc chắn là một trong những phân tích đầu tiên về cách tiếp cận nói trên để lập lịch cho một hệ thống chia sẻ thời gian.

[CK68] “Phương pháp lập lịch máy tính và các biện pháp đối phó của chúng” của Edward G. Coffman và Leonard Kleinrock. AFIPS '68 (Mùa xuân), tháng 4 năm 1968. Giới thiệu ban đầu tuyệt vời và phân tích một số nguyên tắc lập lịch cơ bản.

[J91] “Nghệ thuật Phân tích Hiệu suất Hệ thống Máy tính: Các Kỹ thuật Thực nghiệm Dấu hiệu De, Đo lường, Mô phỏng và Mô hình hóa” của R. Jain. Interscience, New Yorx, tháng 4 năm 1991.
Văn bản tiêu chuẩn về đo lường hệ thống máy tính. Một tài liệu tham khảo tuyệt vời cho thư viện của bạn, chắc chắn.

[O45] "Animal Farm" của George Orwell. Secker và Warburg (London), 1945. Một cuốn sách ngụ ngôn tuyệt vời nhưng buồn bã về quyền lực và sự thối nát của nó. Một số người nói rằng đó là một bài phê bình về Stalin và thời kỳ Stalin trước Thế chiến thứ hai ở Liên Xô; chúng tôi nói đó là một sự phê phán về những con lợn.

[PV56] “Sửa chữa máy như một vấn đề về hàng đợi ưu tiên” của Thomas E. Phipps Jr., WR
Van Voorhis. Nghiên cứu hoạt động, 4: 1, trang 76–86, tháng 2 năm 1956. Công việc tiếp theo khái quát hóa cách tiếp cận của SJF để sửa chữa máy móc từ công việc ban đầu của Cobham; cũng công nhận tiện ích của cách tiếp cận STCF trong môi trường như vậy. Cụ thể, “Có một số loại công việc sửa chữa, ... liên quan đến việc tháo dỡ nhiều và che phủ sàn bằng đai ốc và bu lông, chắc chắn không nên bị gián đoạn một khi đã thực hiện; trong những trường hợp khác, sẽ không thể tiếp tục làm một công việc dài nếu một hoặc nhiều công việc ngắn hạn có sẵn (tr.81). ”

[MB91] “Tác động của việc chuyển đổi ngữ cảnh lên hiệu suất bộ nhớ đệm” của Jeffrey C. Mogul, Anita Borg.
ASPLOS, 1991. Một nghiên cứu thú vị về cách hiệu suất bộ nhớ cache có thể bị ảnh hưởng bởi chuyển đổi ngữ cảnh; ít vấn đề hơn trong các hệ thống ngày nay, nơi bộ xử lý đưa ra hàng tỷ lệnh mỗi giây nhưng chuyển mạch ngữ cảnh vẫn xảy ra trong phạm vi thời gian mili giây.

[W15] “Bạn không thể cầm chiếc bánh của mình và ăn nó” của Tác giả: Unknown .. Wikipedia (tính đến tháng 12 năm 2015). http://en.wikipedia.org/wiki/ Bạn không thể cầm bánh và ăn. _ _ _ _ _ _ _
Phần tốt nhất của trang này là đọc tất cả các thành ngữ tương tự từ các ngôn ngữ khác. Ở Tamil, bạn không thể “vừa để ria mép vừa uống nước canh”.

Bài tập về nhà (Mô phỏng)

Chương trình này, Scheduler.py, cho phép bạn xem các trình quản lý lịch biểu khác nhau hoạt động như thế nào theo các chỉ số lập lịch như thời gian phản hồi, thời gian quay vòng và tổng thời gian chờ. Xem README để biết thêm chi tiết.

Câu hỏi

1. Tính toán thời gian phản hồi và thời gian quay vòng khi chạy ba công việc có độ dài 200 với bộ lập lịch SJF và FIFO.
2. Bây giờ làm tương tự nhưng với các công việc có độ dài khác nhau: 100, 200 và 300.
3. Bây giờ làm tương tự, nhưng cũng với bộ lập lịch RR và một lát thời gian của 1.
4. Đối với những loại khối lượng công việc nào thì SJF cung cấp thời gian quay vòng tương tự như FIFO?
5. Đối với những loại khối lượng công việc và độ dài lượng tử nào thì SJF cung cấp thời gian phản hồi giống như RR?
6. Điều gì xảy ra với thời gian phản hồi với SJF khi thời lượng công việc tăng lên? Bạn có thể sử dụng trình mô phỏng để chứng minh xu hướng không?
7. Điều gì xảy ra với thời gian phản hồi với RR là độ dài lượng tử bị gấp khúc? Bạn có thể viết một phương trình cung cấp thời gian tài trợ cho trường hợp xấu nhất, cho N công việc không?