4

Tóm tắt: Quá trình

Trong chương này, chúng ta thảo luận về một trong những điều trừu tượng cơ bản nhất mà HĐH cung cấp cho người dùng: tiến trình. Định nghĩa của một quá trình, thông thường, khá đơn giản: nó là một chương trình đang chạy [V + 65, BH70]. Bản thân chương trình là một thứ vô hồn: nó chỉ nằm đó trên đĩa, một loạt các hướng dẫn (và có thể là một số dữ liệu tĩnh), chờ đợi để bắt đầu hoạt động. Đó là hệ thống hoạt động lấy các byte này và làm cho chúng chạy, chuyển đổi chương trình thành một thứ gì đó hữu ích.

Nó chỉ ra rằng một người thường muốn chạy nhiều hơn một chương trình cùng một lúc; ví dụ: xem xét máy tính để bàn hoặc máy tính xách tay của bạn, nơi bạn có thể muốn chạy trình duyệt web, chương trình thư, trò chơi, trình phát nhạc, v.v. Trên thực tế, một hệ thống điển hình dường như đang chạy hàng chục hoặc thậm chí hàng trăm quy trình cùng một lúc. Làm như vậy giúp hệ thống dễ sử dụng, vì người ta không bao giờ cần quan tâm đến việc có CPU hay không; một người chỉ đơn giản là chạy các chương trình. Do đó, thách thức của chúng tôi:

SỰ CỐ GẮNG CỦA VẤN ĐỀ: LÀM THẾ NÀO ĐỂ CUNG CẤP ẢNH HƯỞNG CỦA NHIỀU CPUS ?

Mặc dù chỉ có một số CPU vật lý khả dụng, nhưng làm thế nào để Hệ điều hành cung cấp ảo tưởng về một nguồn cung cấp gần như vô tận các CPU nói trên?

Hệ điều hành tạo ra ảo giác này bằng cách ảo hóa CPU. Bằng cách chạy một quá trình, sau đó dừng nó và chạy một quá trình khác, v.v., Hệ điều hành có thể tạo ra ảo tưởng rằng có nhiều CPU ảo tồn tại trong khi thực tế chỉ có một (hoặc một vài) CPU vật lý. Kỹ thuật cơ bản này, được gọi là chia sẻ thời gian của CPU, cho phép người dùng chạy nhiều quá trình đồng thời tùy thích; chi phí tiềm năng là hiệu suất, vì mỗi cái sẽ chạy chậm hơn nếu (các) CPU phải được chia sẻ.

Để thực hiện ảo hóa CPU và để triển khai nó tốt, hệ điều hành sẽ cần cả một số máy móc cấp thấp và một số cấp cao trong kỹ thuật viễn thông. Chúng tôi gọi là các cơ chế máy móc cấp thấp; cơ chế là các phương thức hoặc giao thức cấp thấp thực hiện một phần chức năng cần thiết. Ví dụ: sau này chúng ta sẽ học cách triển khai một ngữ cảnh

MẸO: SỬ DỤNG CHIA SẺ THỜI GIAN (VÀ CHIA SẺ KHÔNG GIAN)

Chia sẻ thời gian là một kỹ thuật cơ bản được sử dụng bởi một hệ điều hành để chia sẻ tài nguyên. Bằng cách cho phép tài nguyên được sử dụng trong một thời gian ngắn bởi một thực thể, sau đó là một thời gian ngắn cho thực thể khác, v.v., tài nguyên được đề cập (ví dụ: CPU hoặc một liên kết mạng) có thể được nhiều người chia sẻ. Đối tác của chia sẻ thời gian là chia sẻ không gian, trong đó tài nguyên được chia (trong không gian) cho những người muốn sử dụng nó. Ví dụ, không gian đĩa tự nhiên là một tài nguyên được chia sẻ không gian; khi một khối được gán cho một tệp, nó thường không được ký vào một tệp khác cho đến khi người dùng xóa tệp gốc.

chuyển đổi, cung cấp cho HĐH khả năng dừng chạy một chương trình và bắt đầu chạy một chương trình khác trên một CPU nhất định; cơ chế chia sẻ thời gian này được sử dụng bởi tất cả các hệ điều hành hiện đại.

Trên đầu các cơ chế này chứa một số thông tin thông minh trong Hệ điều hành, dưới dạng các chính sách. Chính sách là các thuật toán để đưa ra một số loại quyết định trong Hệ điều hành. Ví dụ, với một số chương trình có thể chạy trên CPU, hệ điều hành nên chạy chương trình nào? Chính sách lập lịch trong HĐH sẽ đưa ra quyết định này, có khả năng sử dụng thông tin về lịch sử (ví dụ: chương trình nào đã chạy nhiều hơn trong phút qua?), Kiến thức về khối lượng công việc (ví dụ: loại chương trình nào đang chạy) và chỉ số perfor mance (ví dụ: , hệ thống đang tối ưu hóa hiệu suất tương tác hay thông lượng?) để đưa ra quyết định.

4.1 Tóm tắt: Một quá trình

Sự trừu tượng được cung cấp bởi hệ điều hành của một chương trình đang chạy là thứ mà chúng ta sẽ gọi là một quá trình. Như chúng ta đã nói ở trên, một tiến trình chỉ đơn giản là một chương trình đang chạy; tại bất kỳ thời điểm nào, chúng tôi có thể tóm tắt một quy trình bằng cách lấy một bản kiểm kê các phần khác nhau của hệ thống mà nó truy cập hoặc ảnh hưởng trong quá trình thực thi.

Do đó, để hiểu những gì cấu thành một quy trình, chúng ta phải hiểu trạng thái máy của nó: chương trình có thể đọc hoặc cập nhật những gì khi nó đang chạy.
Tại bất kỳ thời điểm nào, bộ phận nào của máy quan trọng đối với việc thực thi chương trình này?

Một thành phần hiển nhiên của trạng thái máy bao gồm một tiến trình là bộ nhớ của nó. Hướng dẫn nằm trong bộ nhớ; dữ liệu mà pro gram đang chạy đọc và ghi cũng nằm trong bộ nhớ. Vì vậy, vùng nhớ mà tiến trình có thể giải quyết (được gọi là không gian địa chỉ của nó) là một phần của tiến trình.

Ngoài ra, một phần của trạng thái máy của tiến trình là các thanh ghi; nhiều hướng dẫn đọc hoặc cập nhật thanh ghi một cách rõ ràng và do đó rõ ràng chúng rất quan trọng đối với việc thực hiện quy trình.

Lưu ý rằng có một số thanh ghi đặc biệt đặc biệt tạo thành một phần của trạng thái máy này. Ví dụ, bộ đếm chương trình (PC) (đôi khi được gọi là con trỏ lệnh hoặc IP) cho chúng ta biết lệnh nào của pro gram sẽ thực hiện tiếp theo; tương tự như một con trỏ ngăn xếp và khung liên kết

MẸO: CHÍNH SÁCH VÀ CƠ CHẾ RIÊNG Trong nhiều hệ điều hành, một mô hình thiết kế chung là tách các chính sách cấp cao khỏi các cơ chế cấp thấp của chúng [L + 75]. Bạn có thể nghĩ về cơ chế này như cung cấp câu trả lời cho câu hỏi như thế nào về một hệ thống; ví dụ, hệ điều hành thực hiện chuyển đổi ngữ cảnh như thế nào? Chính sách cung cấp câu trả lời cho câu hỏi nào; ví dụ, hệ điều hành nên chạy tiến trình nào ngay bây giờ? Việc tách biệt cả hai cho phép người ta dễ dàng thay đổi các chính sách mà không cần phải suy nghĩ lại về cơ chế và do đó là một dạng mô đun, một nguyên tắc thiết kế phần mềm chung.

con trỏ được sử dụng để quản lý ngăn xếp cho các tham số hàm, ables biến cục bộ và địa chỉ trả về.

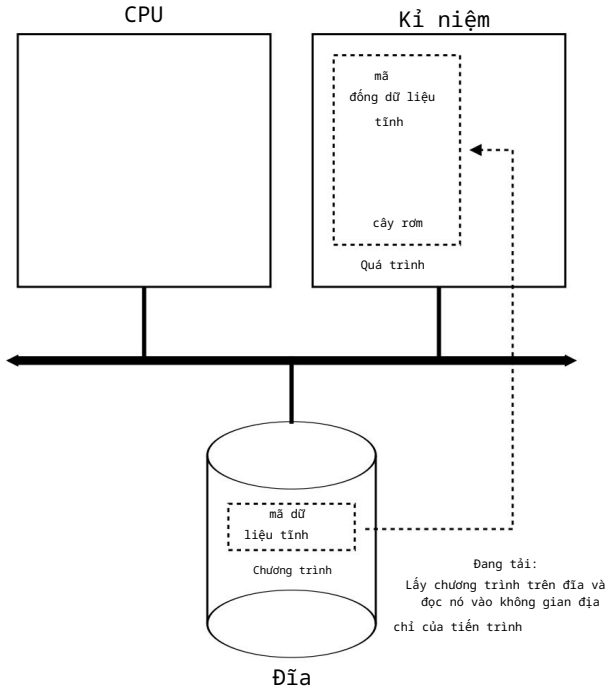
Cuối cùng, các chương trình cũng thường truy cập các thiết bị lưu trữ liên tục. Thông tin I / O như vậy có thể bao gồm danh sách các tệp mà quy trình hiện đang mở.

4.2 Quy trình API

Mặc dù chúng tôi hoãn thảo luận về một API quy trình thực cho đến chương tiếp theo, nhưng ở đây trước tiên chúng tôi đưa ra một số ý tưởng về những gì phải có trong bất kỳ giao diện nào của hệ điều hành. Các API này, ở một số dạng, có sẵn trên bất kỳ hệ điều hành hiện đại nào.

- Tạo: Một hệ điều hành phải bao gồm một số phương pháp để tạo ra các quy trình mới. Khi bạn nhập lệnh vào trình bao hoặc nhấp đúp vào biểu tượng ứng dụng, Hệ điều hành sẽ được gọi để tạo một quy trình mới để chạy chương trình bạn đã chỉ định.
- Phá hủy: Vì có một giao diện để tạo quy trình, các hệ thống cũng cung cấp một giao diện để phá hủy các quy trình một cách cưỡng bức. Tất nhiên, nhiều tiến trình sẽ chạy và chỉ tự thoát ra khi hoàn tất; Tuy nhiên, khi chúng không làm vậy, người dùng có thể muốn giết chúng, và do đó, một biện pháp ngăn chặn quá trình chạy trốn khá hữu ích. • Chờ: Đôi khi rất hữu ích khi đợi một tiến trình ngừng chạy; do đó một số loại giao diện chờ thường được cung cấp. • Kiểm soát Linh tinh: Ngoài việc giết hoặc chờ một quy trình, đôi khi có những kiểm soát khác có thể thực hiện được. Ví dụ: hầu hết các hệ điều hành cung cấp một số loại phương pháp để tạm ngưng một tiến trình (dừng nó chạy trong một thời gian) và sau đó tiếp tục nó (tin tưởng nó đang chạy).

- Trạng thái: Thường có các giao diện để lấy một số thông tin trạng thái về một tiến trình, chẳng hạn như nó đã chạy trong bao lâu hoặc nó đang ở trạng thái nào.



Hình 4.1: Đang tải: Từ chương trình đến quá trình

4.3 Tạo quy trình: Chi tiết hơn một chút

Một bí ẩn mà chúng ta nên tiết lộ một chút là cách các chương trình được chuyển đổi thành các quy trình. Cụ thể, hệ điều hành làm thế nào để khởi động và chạy một chương trình? Việc tạo quy trình thực sự hoạt động như thế nào?

Điều đầu tiên mà Hệ điều hành phải làm để chạy một chương trình là tải mã của nó và bất kỳ dữ liệu tĩnh nào (ví dụ: các biến được khởi tạo) vào bộ nhớ, vào vùng quảng cáo của quy trình. Các chương trình ban đầu nằm trên đĩa (hoặc, trong một số hệ thống hiện đại, SSD dựa trên flash) ở một số loại định dạng thực thi; do đó, quá trình tải một chương trình và dữ liệu tĩnh vào bộ nhớ yêu cầu HĐH đọc các byte đó từ đĩa và đặt chúng vào bộ nhớ ở đâu đó (như trong Hình 4.1).

Trong các hệ điều hành sơ khai (hoặc đơn giản), quá trình tải được thực hiện ngay lập tức, tức là tất cả cùng một lúc trước khi chạy chương trình; hệ điều hành hiện đại thực hiện quá trình một cách lười biếng, tức là chỉ tải các đoạn mã hoặc dữ liệu khi chúng cần thiết trong quá trình thực thi chương trình. Để thực sự hiểu cách hoạt động của việc tải từng đoạn mã và dữ liệu, bạn sẽ phải hiểu thêm về

máy móc phân trang và hoán đổi, các chủ đề mà chúng ta sẽ đề cập trong tương lai khi chúng ta thảo luận về ảo hóa bộ nhớ. Bây giờ, chỉ cần nhớ rằng trước khi chạy bất cứ điều gì, hệ điều hành rõ ràng phải thực hiện một số công việc để đưa các bit chương trình quan trọng từ đĩa vào bộ nhớ.

Sau khi mã và dữ liệu tĩnh được tải vào bộ nhớ, hệ điều hành cần thực hiện một số việc khác trước khi chạy quy trình. Một số ory mem phải được cấp phát cho ngăn xếp thời gian chạy của chương trình (hoặc chỉ ngăn xếp).
Như bạn có thể đã biết, các chương trình C sử dụng ngăn xếp cho các biến cục bộ, tham số hàm và địa chỉ trả về; hệ điều hành phân bổ bộ nhớ này và cung cấp cho quá trình. Hệ điều hành cũng có thể sẽ khởi tạo ngăn xếp bằng các đối số; cụ thể, nó sẽ điền các tham số vào hàm main () , tức là argc và mảng argv .

Hệ điều hành cũng có thể phân bổ một số bộ nhớ cho heap của chương trình. Trong các chương trình C, heap được sử dụng cho dữ liệu được cấp phát động được yêu cầu rõ ràng; chương trình yêu cầu không gian như vậy bằng cách gọi malloc () và giải phóng nó một cách hợp lý bằng cách gọi free (). Heap cần thiết cho các cấu trúc dữ liệu như danh sách được liên kết, bảng băm, cây và các cấu trúc dữ liệu thú vị khác. Lúc đầu đống sẽ nhỏ; khi chương trình chạy và yêu cầu thêm mem ory thông qua API thư viện malloc () , Hệ điều hành có thể tham gia và phân bổ nhiều bộ nhớ hơn cho quá trình để giúp đáp ứng các lệnh gọi như vậy.

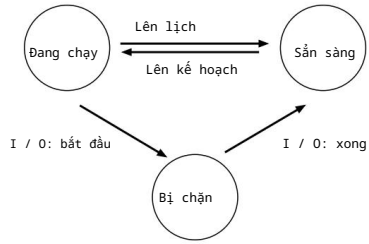
Hệ điều hành cũng sẽ thực hiện một số tác vụ khởi tạo khác, đặc biệt là liên quan đến đầu vào / đầu ra (I / O). Ví dụ, trong hệ thống UNIX , mỗi quá trình theo mặc định có ba bộ mô tả tệp mở, cho đầu vào, đầu ra và lỗi tiêu chuẩn; các bộ mô tả này cho phép các chương trình dễ dàng đọc đầu vào từ thiết bị đầu cuối và in đầu ra ra màn hình. Chúng ta sẽ tìm hiểu thêm về I / O, mã mô tả tệp và những thứ tương tự trong phần ba của cuốn sách về tính bền bỉ.

Bằng cách tải mã và dữ liệu tĩnh vào bộ nhớ, bằng cách tạo và sắp xếp ngăn xếp, và bằng cách thực hiện các công việc khác liên quan đến thiết lập I / O, hệ điều hành giờ đây (cuối cùng) đã tạo tiền đề cho việc thực thi chương trình. Do đó, nó có một nhiệm vụ cuối cùng: khởi động chương trình đang chạy tại điểm nhập, cụ thể là main (). Bằng cách chuyển sang quy trình chính () (thông qua một cơ chế chuyên biệt mà chúng ta sẽ thảo luận ở chương sau), Hệ điều hành chuyển quyền kiểm soát CPU sang quy trình mới được tạo và do đó chương trình bắt đầu thực thi.

4.4 Các trạng thái quá trình

Bây giờ chúng ta đã có một số ý tưởng về quy trình là gì (mặc dù chúng tôi sẽ tiếp tục hoàn thiện khái niệm này) và (đại khái) cách nó được tạo ra, chúng ta hãy nói về các trạng thái khác nhau mà một quy trình có thể ở một thời điểm nhất định. Khái niệm rằng một quá trình có thể ở một trong những trạng thái này đã nảy sinh trong các hệ thống máy tính thời kỳ đầu [DV66, V + 65]. Theo quan điểm đơn giản, một quy trình có thể ở một trong ba trạng thái:

- Đang chạy: Ở trạng thái đang chạy, một tiến trình đang chạy trên một bộ xử lý. Điều này có nghĩa là nó đang thực thi các hướng dẫn.
- Sẵn sàng: Ở trạng thái sẵn sàng, một tiến trình đã sẵn sàng để chạy nhưng vì lý do nào đó mà Hệ điều hành đã chọn không chạy nó vào thời điểm nhất định này.



Hình 4.2: Quy trình: Chuyển đổi trạng thái

- Bị chặn: Ở trạng thái bị chặn, một quá trình đã thực hiện một số loại hoạt động khiến nó không sẵn sàng chạy cho đến khi một số sự kiện khác diễn ra. Một ví dụ phổ biến: khi một quy trình bắt đầu I / O yêu cầu đến một đĩa, nó bị chặn và do đó một số quá trình khác có thể sử dụng bộ xử lý.

Nếu chúng ta ánh xạ các trạng thái này thành biểu đồ, chúng ta sẽ đi đến di agram trong Hình 4.2. Như bạn có thể thấy trong sơ đồ, một quá trình có thể di chuyển giữa các trạng thái sẵn sàng và đang chạy theo quyết định của HĐH. Được chuyển từ trạng thái sẵn sàng sang đang chạy có nghĩa là quá trình đã được lên lịch; được chuyển từ trạng thái đang chạy sang sẵn sàng có nghĩa là quá trình đã hẹn trước. Khi một quá trình đã bị chặn (ví dụ: bằng cách bắt đầu một Hoạt động I / O), hệ điều hành sẽ giữ nó như vậy cho đến khi một số sự kiện xảy ra (ví dụ: I / O hoàn thành); tại thời điểm đó, quy trình lại chuyển sang trạng thái sẵn sàng (và có thể ngay lập tức chạy lại, nếu hệ điều hành quyết định như vậy).

Hãy xem một ví dụ về cách hai quy trình có thể chuyển đổi qua một số trạng thái này. Đầu tiên, hãy tưởng tượng hai quy trình đang chạy, mỗi quy trình chỉ sử dụng CPU (chúng không có I / O). Trong trường hợp này, một dấu vết về trạng thái của mỗi quy trình có thể giống như thế này (Hình 4.3).

Tiến trình thời gian0	Tiến trình1	Ghi chú
1	Sẵn sàng chạy	
2	Sẵn sàng chạy	
3	Sẵn sàng chạy	
4	Đang chạy	Tiến trình sẵn sàng0 hiện đã hoàn tất
5	-	Đang chạy
6	-	Đang chạy
7	-	Đang chạy
8	-	Đang chạy Process1 hiện đã hoàn tất

Hình 4.3: Trạng thái quy trình theo dõi: Chỉ CPU

Tiến trình thời gian0	Tiến trình1	Ghi chú
1	Sẵn sàng chạy	
2	Sẵn sàng chạy	
3	Tiến trình sẵn sàng chạy 0 bắt đầu I / O	
4	Quy trình chạy bị chặn0 bị chặn, Chạy bị chặn để Process1 chạy	
5 6	Đang chạy bị chặn	
7	Sẵn sàng chạy	I / O đã xong
“	Quá trình chạy sẵn sàng1 hiện đã hoàn tất	
9	Đang chạy -	
10	Đang chạy -	Process0 hiện đã hoàn thành

Hình 4.4: Trạng thái quy trình theo dõi: CPU và I / O

Trong ví dụ tiếp theo này, quy trình đầu tiên đưa ra I / O sau khi chạy cho thỉnh thoảng. Tại thời điểm đó, quá trình này bị chặn, đưa ra quá trình khác một cơ hội để chạy. Hình 4.4 cho thấy một dấu vết của kịch bản này.

Cụ thể hơn, Process0 bắt đầu I / O và bị chặn chờ quá trình hoàn tất; các quy trình bị chặn, ví dụ, khi đọc từ đĩa hoặc chờ một gói từ mạng. Hệ điều hành nhận biết Process0 không sử dụng CPU và bắt đầu chạy Process1. Trong khi

Process1 đang chạy, I / O hoàn tất, chuyển Process0 trở lại trạng thái sẵn sàng. Cuối cùng, Process1 kết thúc, và Process0 chạy và sau đó là xong.

Lưu ý rằng có nhiều quyết định mà Hệ điều hành phải đưa ra, ngay cả trong điều này ví dụ đơn giản. Đầu tiên, hệ thống phải quyết định chạy Process1 trong khi Process0 cấp I / O; làm như vậy sẽ cải thiện việc sử dụng tài nguyên bằng cách giữ cho CPU luôn bận rộn. Thứ hai, hệ thống quyết định không chuyển trở lại Process0 khi I / O của nó hoàn thành; không rõ liệu đây có phải là một deci sion tốt hay không. Bạn nghĩ sao? Những loại quyết định này được thực hiện bởi Lập lịch hệ điều hành , một chủ đề chúng ta sẽ thảo luận trong một vài chương trong tương lai.

4.5 Cấu trúc dữ liệu

Hệ điều hành là một chương trình và giống như bất kỳ chương trình nào, nó có một số cấu trúc dữ liệu quan trọng để theo dõi các phần thông tin liên quan khác nhau. Để theo dõi trạng thái của mỗi quy trình, chẳng hạn, hệ điều hành có thể sẽ giữ một số loại danh sách ngừng chuyên nghiệp cho tất cả các quy trình đã sẵn sàng và một số thông tin bổ sung để theo dõi quy trình nào hiện đang chạy. Hệ điều hành cũng phải theo dõi, theo một cách nào đó, các quy trình bị chặn; khi một sự kiện I / O hoàn tất, hệ điều hành nên đảm bảo đánh thức đúng quy trình và sẵn sàng chạy lại.

Hình 4.5 cho thấy loại thông tin mà hệ điều hành cần theo dõi mỗi tiến trình trong nhân xv6 [CK + 08]. Các cấu trúc quy trình tương tự tồn tại trong hệ điều hành "thực" như Linux, Mac OS X hoặc Windows; nhìn chúng lên và xem chúng phức tạp hơn bao nhiêu.

Từ hình này, bạn có thể thấy một vài thông tin quan trọng mà hệ điều hành theo dõi về một quá trình. Bối cảnh đăng ký sẽ được giữ, cho một

```
// các thanh ghi xv6 sẽ lưu và khôi phục
// để dừng và sau đó khởi động lại quá trình
cấu trúc ngữ cảnh {
   int eip;
   int esp;
   int ebx;
   int ecx;
   int edx;
   int esi;
   int edi;
   int ebp;
};

// các trạng thái khác nhau mà một tiến trình có thể ở
enum proc_state {UNUSED, EMBRYO, SLEEPING,
                            RUNNABLE, RUNNING, ZOMBIE};

// thông tin xv6 theo dõi về mỗi quá trình
// bao gồm ngữ cảnh và trạng thái đăng ký của nó
struct proc {
   char * mem;                                  // Bắt đầu bộ nhớ tiến trình
   uint sz; char                                // Kích thước của bộ nhớ tiến trình
   * kstack;                                    // Dưới cùng của ngăn xếp hạt nhân
                                                // cho quá trình này
   trạng thái enum proc_state; int              // Trạng thái xử lý
   pid; // Xử lý ID
   struct proc * cha; // If! Zero,              // Tiến trình chính
   void * chan; //                 đang ngủ trên chan
   int bị giết; tậpIf! Zero, đã bị giết
   tin struct * ofile [NOFILE]; // Mở tập tin
   struct inode * cwd; cấu                      // Thư mục hiện tại
   trúc ngữ cảnh ngữ cảnh; struct               // Chuyển sang đây để chạy quá trình
   trapframe * tf;                              // Khung bẫy cho
                                                // ngắt hiện tại
};
```

Hình 4.5: Cấu trúc xv6 Proc

quá trình dừng, nội dung của sổ đăng ký của nó. Khi một quá trình bị dừng, các thanh ghi của nó sẽ được lưu vào vị trí bộ nhớ này; bằng cách khôi phục các bộ đếm reg này (tức là, đặt các giá trị của chúng trở lại các thanh ghi vật lý thực tế), Hệ điều hành có thể tiếp tục chạy quá trình. Chúng ta sẽ tìm hiểu thêm về kỹ thuật này được biết đến như một chuyển đổi ngữ cảnh trong các chương trong tương lai.

Bạn cũng có thể thấy từ hình rằng có một số trạng thái khác mà điểm chuyên nghiệp có thể ở, ngoài việc chạy, sẵn sàng và bị chặn. Đôi khi một hệ thống sẽ có **trạng thái** ban đầu mà tiến trình đang ở khi nó đang được tạo. Ngoài ra, một quá trình có thể được đặt ở **trạng thái** cuối cùng , nơi nó đã thoát nhưng

BÊN TRONG : CẤU TRÚC DỮ LIỆU - DANH SÁCH QUY TRÌNH

Hệ điều hành có đầy đủ các cấu trúc dữ liệu quan trọng khác nhau mà chúng ta sẽ thảo luận trong các ghi chú này. Danh sách quy trình (còn được gọi là danh sách tác vụ) là cấu trúc đầu tiên như vậy. Nó là một trong những hệ điều hành đơn giản hơn, nhưng chắc chắn bất kỳ hệ điều hành nào có khả năng chạy nhiều chương trình cùng một lúc sẽ có cấu trúc tương tự như cấu trúc này để theo dõi tất cả các chương trình đang chạy trong hệ thống. Đôi khi mọi người đề cập đến cấu trúc cấu trúc riêng lẻ lưu trữ thông tin về một quy trình dưới dạng Khối điều khiển quy trình (PCB), một cách nói hoa mỹ để nói về cấu trúc C chứa thông tin về mỗi quy trình (đôi khi còn được gọi là bộ mô tả quy trình).

vẫn chưa được dọn dẹp (trong các hệ thống dựa trên UNIX, trạng thái này được gọi là trạng thái thây ma[1]). Trạng thái cuối cùng này có thể hữu ích vì nó cho phép các quy trình khác (thường là quy trình mẹ tạo ra quy trình) kiểm tra mã trả về của quy trình và xem liệu quy trình vừa kết thúc có được thực thi thành công hay không (thông thường, các chương trình trả về 0 trong hệ thống dựa trên UNIX khi họ đã hoàn thành một nhiệm vụ thành công và khác 0). Khi vây được băm, cha mẹ sẽ thực hiện một cuộc gọi cuối cùng (ví dụ: wait ()) để đợi hoàn thành phần tử con và cũng để chỉ ra cho OS biết rằng nó có thể dọn dẹp mọi cấu trúc dữ liệu liên quan được đề cập đến quá trình.

4.6 Tóm tắt

Chúng tôi đã giới thiệu phần trừu tượng cơ bản nhất của HĐH: quy trình. Nó khá đơn giản được xem như một chương trình đang chạy. Với quan điểm khái niệm này, bây giờ chúng ta sẽ chuyển sang phần thực tế: các cơ chế cấp thấp cần thiết để thực hiện các quy trình và các cơ chế chính trị cấp cao hơn cần thiết để lập lịch trình cho chúng một cách thông minh. Bằng cách kết hợp các anisms và chính sách mech, chúng ta sẽ xây dựng hiểu biết của mình về cách một hệ thống oper ating ảo hóa CPU.

[1] Có, trạng thái zombie. Cũng giống như những thây ma thực sự, những thây ma này tương đối dễ tiêu diệt. Tuy nhiên, các kỹ thuật khác nhau thường được khuyến khích.

ASIDE: ĐIỀU KHOẢN CHÍNH CỦA QUY TRÌNH

- Tiến trình là phần tóm tắt hệ điều hành chính của một chương trình đang chạy. Tại bất kỳ thời điểm nào, quá trình này có thể được mô tả bằng trạng thái của nó: lều của bộ nhớ trong không gian địa chỉ của nó, nội dung của các thanh ghi CPU (bao gồm bộ đếm chương trình và con trỏ ngăn xếp, trong số những thứ khác), và thông tin về I / O (chẳng hạn như các tệp đang mở có thể đọc được hoặc bằng văn bản).
- API quy trình bao gồm các cuộc gọi mà các chương trình có thể thực hiện liên quan đến các điểm chuyên nghiệp. Thông thường, điều này bao gồm tạo, phá hủy và các cuộc gọi hoàn toàn sử dụng khác.
- Quy trình tồn tại ở một trong nhiều trạng thái quy trình khác nhau, bao gồm đang chạy, sẵn sàng chạy và bị chặn. Các sự kiện khác nhau (ví dụ: nhận được đã lên lịch hoặc lên lịch, hoặc đợi I / O hoàn thành) chuyển một quy trình từ một trong những trạng thái này sang trạng thái khác.
- Một danh sách quy trình chứa thông tin về tất cả các quy trình trong hệ thống. Mỗi mục nhập được tìm thấy trong cái mà đôi khi được gọi là quá trình khối điều khiển (PCB), thực sự chỉ là một cấu trúc chứa thông tin về một quy trình cụ thể.

Người giới thiệu

[BH70] “Hạt nhân của một hệ thống đa chương trình” của Per Brinch Hansen. Truyền thông tin của ACM, Tập 13: 4, tháng 4 năm 1970. Bài báo này giới thiệu một trong những kênh vi mô đầu tiên trong lịch sử hệ điều hành, được gọi là Hạt nhân. Ý tưởng về các hệ thống nhỏ hơn, tối giản hơn là một chủ đề lặp đi lặp lại trong lịch sử hệ điều hành; tất cả bắt đầu với công việc của Brinch Hansen được mô tả ở đây.

[CK + 08] “Hệ điều hành xv6” của Russ Cox, Frans Kaashoek, Robert Morris, Nickolai Zeldovich. Từ: https://github.com/mit-pdos/xv6-public. Hệ điều hành thực tế và thú vị nhất trên thế giới. Tải xuống và chơi với nó để tìm hiểu thêm về chi tiết cách hệ điều hành thực sự hoạt động. Chúng tôi đang sử dụng phiên bản cũ hơn (2012-01-30-1-g1c41342) và do đó một số ví dụ trong sách có thể không khớp với phiên bản mới nhất trong nguồn.

[DV66] “Ngữ nghĩa lập trình cho các phép tính đa chương trình” của Jack B. Dennis, Earl C. Van Horn. Thông tin liên lạc của ACM, Tập 9, Số 3, tháng 3 năm 1966. Bài báo này đã định nghĩa nhiều thuật ngữ và khái niệm ban đầu xung quanh việc xây dựng các hệ thống đa chương trình.

[L + 75] “Phân tách chính sách / cơ chế trong Hydra” của R. Levin, E. Cohen, W. Corwin, F. Pollack, W. Wulf. SOSP '75, Austin, Texas, tháng 11 năm 1975. Một bài báo ban đầu về cách cấu trúc các hệ thống mở trong một hệ điều hành nghiên cứu được gọi là Hydra. Mặc dù Hydra chưa bao giờ trở thành một hệ điều hành chính thống, nhưng một số ý tưởng của nó đã ảnh hưởng đến các nhà thiết kế hệ điều hành.

[V + 65] “Structure of the Multics Supervisor” của VA Vyssotsky, FJ Corbato, RM Graham.
Hội nghị Máy tính Chung mùa thu, 1965. Một bài báo ban đầu về Đa phương tiện, mô tả nhiều ý tưởng và thuật ngữ cơ bản mà chúng ta tìm thấy trong các hệ thống hiện đại. Một số tầm nhìn đằng sau điện toán như một tiện ích cuối cùng đã được hiện thực hóa trong các hệ thống đám mây hiện đại.

Bài tập về nhà (Mô phỏng)

Chương trình này, process-run.py, cho phép bạn xem trạng thái quy trình thay đổi như thế nào khi chương trình chạy và sử dụng CPU (ví dụ: thực hiện thêm lệnh) hoặc thực hiện I / O (ví dụ: gửi yêu cầu đến đĩa và đợi nó để hoàn thành). Xem README để biết thêm chi tiết.

Câu hỏi 1.

Chạy process-run.py với các cờ sau: -l 5: 100,5: 100.
Việc sử dụng CPU nên là bao nhiêu (ví dụ: phần trăm thời gian CPU được sử dụng?) Tại sao bạn biết điều này? Sử dụng các cờ -c và -p để xem bạn có đúng không.

2. Bây giờ hãy chạy với các cờ sau: ./process-run.py -l 4: 100,1: 0.
Các cờ này chỉ định một quy trình với 4 hướng dẫn (tất cả đều để sử dụng CPU) và một quy trình chỉ đưa ra một I / O và đợi nó được thực hiện.
Mất bao lâu để hoàn thành cả hai quy trình? Sử dụng -c và -p để tìm hiểu xem bạn có đúng không.

3. Chuyển thứ tự của các quá trình: -l 1: 0,4: 100. Điều gì xảy ra bây giờ?
Chuyển đổi thứ tự có quan trọng không? Tại sao? (Như mọi khi, hãy sử dụng -c và -p để xem bạn có đúng không)

4. Bây giờ chúng ta sẽ khám phá một số cờ khác. Một lá cờ quan trọng là -S, xác định cách hệ thống phản ứng khi một quá trình bị kiện một I / O. Với cờ được đặt thành SWITCH ON END, hệ thống sẽ KHÔNG chuyển sang một quá trình khác trong khi một quá trình đang thực hiện I / O, thay vào đó là đợi cho đến khi quá trình kết thúc hoàn toàn. Điều gì xảy ra khi bạn chạy hai quy trình sau (-l 1: 0,4: 100 -c -S SWITCH ON END), một làm I / O và một làm việc CPU? – –

5. Bây giờ, chạy các quy trình tương tự, nhưng với hành vi chuyển đổi được đặt để chuyển sang quy trình khác bất cứ khi nào một quy trình đang CHỜ I / O (-l 1: 0,4: 100 -c -S SWITCH ON IO). Điều gì xảy ra bây giờ? Sử dụng -c và -p để xác nhận rằng bạn đúng.

6. Một hành vi quan trọng khác là phải làm gì khi I / O com hoạt động kém hiệu quả. Với -I IO RUN LATER, khi I / O hoàn tất, điểm chuyên nghiệp đã cấp cho nó không nhất thiết phải chạy ngay lập tức; đúng hơn, bất cứ thứ gì đang chạy vào thời điểm đó vẫn tiếp tục chạy. Điều gì xảy ra khi bạn chạy kết hợp các quy trình này? (Chạy ./process-run.py -l 3: 0,5: 100,5: 100,5: 100 -S CHUYỂN ĐỔI IO-IO CHẠY SAU -c -p) Tài nguyên hệ thống có được sử dụng hiệu quả không?

7. Bây giờ hãy chạy các quy trình tương tự, nhưng với bộ IO RUN NGAY LẬP TỨC , bộ này sẽ ngay lập tức chạy quy trình đã cấp I / O. Hành vi này khác nhau như thế nào? Tại sao việc chạy một quy trình vừa bắt đầu yêu cầu I / O lại có thể là một ý kiến hay?

8. Bây giờ hãy chạy với một số quy trình được tạo ngẫu nhiên: -s 1 -l 3: 50,3: 50 hoặc -s 2 -l 3: 50,3: 50 hoặc -s 3 -l 3: 50,3: 50. Hãy xem liệu bạn có thể đoán được dấu vết sẽ xuất hiện như thế nào không. Điều gì xảy ra khi bạn sử dụng cờ -I IO RUN NGAY LẬP TỨC so với IO IO RUN L SAU? Những cây bút nào hữu hiệu khi bạn sử dụng -S CHUYỂN ĐỔI IO so với -S CHUYỂN ĐỔI BẬT KẾT THÚC? – –