6

Cơ chế: Thực thi trực tiếp có giới hạn

Để ảo hóa CPU, hệ điều hành cần bằng cách nào đó chia sẻ CPU vật lý giữa nhiều công việc chạy dường như cùng một lúc. Ý tưởng cơ bản rất đơn giản: chạy một quá trình trong một thời gian ngắn, sau đó chạy một quá trình khác, v.v. Theo thời gian chia sẻ CPU theo cách này, ảo hóa sẽ đạt được.

Tuy nhiên, có một số thách thức trong việc xây dựng bộ máy ảo hóa như vậy. Đầu tiên là hiệu suất: làm thế nào chúng ta có thể triển khai virtualiza tion mà không cần thêm chi phí quá mức vào hệ thống? Thứ hai là kiểm soát: làm thế nào chúng ta có thể chạy các quy trình một cách hiệu quả trong khi vẫn giữ được quyền kiểm soát đối với CPU? Kiểm soát đặc biệt quan trọng đối với HĐH, vì nó phụ trách các nguồn lực; nếu không có sự kiểm soát, một quá trình có thể đơn giản chạy mãi mãi và chiếm quyền điều khiển máy hoặc truy cập thông tin mà nó không được phép truy cập. Do đó, đạt được hiệu suất cao trong khi duy trì quyền kiểm soát là một trong những thách thức trọng tâm trong việc xây dựng một hệ điều hành.

CRUX :

CÁCH XỬ LÝ HIỆU QUẢ CPU BẰNG ĐIỀU KHIỂN

Hệ điều hành phải ảo hóa CPU một cách hiệu quả trong khi vẫn duy trì quyền kiểm soát hệ thống. Để làm như vậy, cả phần cứng và hệ điều hành đều được hỗ trợ. Hệ điều hành thường sẽ sử dụng một chút hỗ trợ thiết bị cứng cẩn thận để hoàn thành công việc của mình một cách hiệu quả.

6.1 Kỹ thuật cơ bản: Thực hiện trực tiếp có giới hạn

Để làm cho một chương trình chạy nhanh như người ta có thể mong đợi, không ngạc nhiên khi các nhà phát triển hệ điều hành đã đưa ra một kỹ thuật mà chúng tôi gọi là thực thi trực tiếp có giới hạn. Phần "thực thi trực tiếp" của ý tưởng rất đơn giản: chỉ cần chạy chương trình trực tiếp trên CPU. Do đó, khi hệ điều hành muốn khởi động một chương trình chuyên nghiệp đang chạy, nó sẽ tạo một mục nhập quy trình cho nó trong danh sách quy trình, cấp phát một số bộ nhớ cho nó, tải mã chương trình vào bộ nhớ (từ đĩa), lo ghi điểm vào của nó (tức là, chính () thói quen hoặc một cái gì đó tương tự), nhảy

Hệ điều hành	Chương trình
Tạo mục nhập cho danh sách quy trình	
Cấp phát bộ nhớ cho chương trình	
Tải chương trình vào bộ nhớ	
Thiết lập ngăn xếp với argc / argv	
Xóa sổ đăng ký	
Thực thi cuộc gọi main ()	
	Chạy main ()
	Thực thi trả về từ chính
Bộ nhớ miễn phí của quá trình	
Xóa khỏi danh sách quy trình	

Hình 6.1: Giao thức thực thi trực tiếp (Không giới hạn)

vào nó và bắt đầu chạy mã của người dùng. Hình 6.1 cho thấy giao thức thực thi di trực tràng cơ bản này (chưa có bất kỳ giới hạn nào), sử dụng lệnh gọi bình thường và quay trở lại để nhảy đến main () của chương trình và sau đó quay trở lại hạt nhân.

Nghe có vẻ đơn giản phải không? Nhưng cách tiếp cận này dẫn đến một số vấn đề trong nhiệm vụ ảo hóa CPU của chúng tôi. Đầu tiên rất đơn giản: nếu chúng ta chỉ chạy một chương trình, làm thế nào hệ điều hành có thể đảm bảo rằng chương trình đó không làm bất cứ điều gì mà chúng ta không muốn nó làm, trong khi vẫn chạy nó một cách hiệu quả? Thứ hai: khi chúng ta đang chạy một tiến trình, làm thế nào để hệ điều hành ngăn nó chạy và chuyển sang một tiến trình khác, do đó thực hiện chia sẻ thời gian mà chúng ta yêu cầu để ảo hóa CPU?

Khi trả lời những câu hỏi dưới đây, chúng ta sẽ hiểu rõ hơn về những gì cần thiết để ảo hóa CPU. Khi phát triển các kỹ thuật này, chúng ta cũng sẽ xem phần "giới hạn" của tên phát sinh từ đâu; không có giới hạn về việc chạy các chương trình, hệ điều hành sẽ không kiểm soát được bất cứ thứ gì và do đó sẽ “chỉ là một thư viện” - một tình trạng rất đáng buồn đối với một hệ điều hành đầy tham vọng!

6.2 Vấn đề # 1: Các hoạt động bị hạn chế

Thực hiện trực tiếp có lợi thế rõ ràng là nhanh chóng; chương trình chạy nguyên bản trên CPU phần cứng và do đó thực thi nhanh như người ta mong đợi. Nhưng việc chạy trên CPU dẫn đến một vấn đề: điều gì sẽ xảy ra nếu quá trình muốn thực hiện một số loại hoạt động bị hạn chế, chẳng hạn như đưa ra yêu cầu I / O cho đĩa hoặc giành quyền truy cập vào nhiều tài nguyên hệ thống hơn như CPU hoặc bộ nhớ?

THE CRUX: CÁCH THỰC HIỆN CÁC HOẠT ĐỘNG HẠN CHẾ

Một quy trình phải có khả năng thực hiện I / O và một số hoạt động hạn chế khác, nhưng không cho quy trình kiểm soát hoàn toàn hệ thống.
Làm thế nào hệ điều hành và phần cứng có thể hoạt động cùng nhau để làm như vậy?

BÊN TRONG: TẠI SAO CÁC CUỘC GỌI HỆ THỐNG NHÌN NHƯ THỦ TỤC CUỘC GỌI

Bạn có thể thắc mắc tại sao một lệnh gọi đến một lệnh gọi hệ thống, chẳng hạn như open () hoặc read (), trông giống hệt như một lệnh gọi thủ tục điển hình trong C; nghĩa là, nếu nó trông giống như một cuộc gọi thủ tục, làm thế nào hệ thống biết đó là một cuộc gọi hệ thống và thực hiện tất cả những thứ phù hợp? Lý do đơn giản: nó là một cuộc gọi thủ tục, nhưng ẩn bên trong lời gọi thủ tục đó là lệnh bẫy nổi tiếng. Cụ thể hơn, khi bạn gọi open () (ví dụ), bạn đang thực hiện một lệnh gọi thủ tục vào thư viện C. Trong đó, cho dù đối với lệnh open () hay bất kỳ lệnh gọi hệ thống nào khác được cung cấp, thư viện sử dụng quy ước gọi đã thống nhất với hạt nhân để đặt các đối số để mở () ở các vị trí nổi tiếng (ví dụ: trên ngăn xếp, hoặc trong các thanh ghi cụ thể), đặt số gọi hệ thống vào một vị trí đã biết (một lần nữa, vào ngăn xếp hoặc một thanh ghi), rồi thực hiện lệnh bẫy đã nói ở trên. Mã trong thư viện sau khi giải nén bẫy sẽ trả về các giá trị và trả lại quyền điều khiển cho chương trình đã thực hiện lệnh gọi hệ thống. Do đó, các phần của thư viện C thực hiện lệnh gọi hệ thống được mã hóa thủ công trong assembly, vì chúng cần tuân theo quy ước cẩn thận để xử lý các đối số và trả về giá trị cor trực tiếp, cũng như thực hiện lệnh bẫy dành riêng cho phần cứng. Và bây giờ bạn biết tại sao cá nhân bạn không cần phải viết mã lắp ráp để bẫy vào một hệ điều hành; ai đó đã viết bản lắp ráp đó cho bạn.

Một cách tiếp cận đơn giản là để bất kỳ quy trình nào làm bất cứ điều gì nó muốn về I / O và các hoạt động liên quan khác. Tuy nhiên, làm như vậy sẽ ngăn cản việc xây dựng nhiều loại hệ thống được mong muốn. Ví dụ: nếu chúng ta muốn xây dựng một hệ thống tệp kiểm tra quyền trước khi cấp quyền truy cập vào tệp, chúng ta không thể đơn giản để bất kỳ quá trình người dùng nào cấp I / Os cho đĩa; nếu chúng ta làm vậy, một tiến trình có thể chỉ đọc hoặc ghi toàn bộ đĩa và do đó tất cả các biện pháp bảo vệ sẽ bị mất.

Do đó, cách tiếp cận mà chúng tôi thực hiện là giới thiệu một chế độ xử lý mới, được gọi là chế độ người dùng; mã chạy ở chế độ người dùng bị hạn chế về những gì nó có thể làm. Ví dụ: khi chạy ở chế độ người dùng, một quy trình không thể đưa ra các yêu cầu I / O; làm như vậy sẽ dẫn đến việc bộ xử lý đưa ra một ngoại lệ; hệ điều hành sau đó có thể sẽ giết quá trình.

Ngược lại với chế độ người dùng là chế độ hạt nhân, mà hệ điều hành (hoặc hạt nhân) chạy trong. Trong chế độ này, mã chạy có thể làm những gì nó thích, bao gồm các hoạt động đặc quyền như đưa ra yêu cầu I / O và thực hiện tất cả các loại hạn chế hướng dẫn.

Tuy nhiên, chúng tôi vẫn còn một thách thức: người dùng chuyên nghiệp nên làm gì khi họ muốn thực hiện một số loại hoạt động đặc quyền, chẳng hạn như đọc từ đĩa? Để thực hiện điều này, hầu như tất cả các thiết bị cứng hiện đại đều cung cấp khả năng cho các chương trình người dùng thực hiện lệnh gọi hệ thống. Tiên phong trên các máy cổ điển như Atlas [K + 61, L78], các lệnh gọi hệ thống cho phép hạt nhân hiển thị cẩn thận các phần chức năng chính nhất định cho các chương trình của người dùng, chẳng hạn như truy cập vào hệ thống tệp, tạo và hủy các quy trình ing, giao tiếp với các quy trình khác và phân bổ nhiều hơn

MẸO: SỬ DỤNG CHUYỂN GIAO ĐIỀU KHIỂN ĐƯỢC BẢO VỆ Phần cứng hỗ trợ HĐH bằng cách cung cấp các chế độ thực thi khác nhau.
Ở chế độ người dùng, các ứng dụng không có toàn quyền truy cập vào tài nguyên phần cứng. Trong chế độ hạt nhân, hệ điều hành có quyền truy cập vào toàn bộ tài nguyên của máy.
Các hướng dẫn đặc biệt để bẫy vào hạt nhân và trả về từ bẫy trở lại các chương trình chế độ người dùng, cũng như các hướng dẫn cho phép HĐH thông báo cho phần cứng biết vị trí của bảng bẫy trong bộ nhớ.

kỉ niệm. Hầu hết các hệ điều hành cung cấp một vài trăm cuộc gọi (xem tiêu chuẩn POSIX để biết thêm chi tiết [P10]); các hệ thống Unix ban đầu cho thấy một tập hợp con ngắn gọn hơn gồm khoảng 20 lệnh gọi.

Để thực hiện một lệnh gọi hệ thống, một chương trình phải thực hiện một lệnh bẫy đặc biệt . Lệnh này đồng thời nhảy vào hạt nhân và nâng mức đặc quyền lên chế độ hạt nhân; khi ở trong hạt nhân, hệ thống giờ đây có thể ở mỗi dạng bất kỳ hoạt động đặc quyền nào cần thiết (nếu được phép), và do đó thực hiện công việc cần thiết cho quá trình gọi. Khi hoàn tất, HĐH gọi một lệnh trả về từ bẫy đặc biệt , như bạn có thể mong đợi, sẽ quay trở lại chương trình người dùng đang gọi trong khi đồng thời giảm mức privi lege trở lại chế độ người dùng.

Phần cứng cần phải cẩn thận một chút khi thực hiện một bẫy, trong đó nó phải đảm bảo lưu đủ số đăng ký của người gọi để có thể trả về chính xác khi HĐH đưa ra lệnh trả về từ bẫy.
Ví dụ, trên x86, bộ xử lý sẽ đẩy bộ đếm chương trình, cờ và một số thanh ghi khác vào một ngăn xếp hạt nhân cho mỗi quá trình; bẫy trả về sẽ bật các giá trị này ra khỏi ngăn xếp và tiếp tục thực thi chương trình chế độ người dùng (xem hướng dẫn sử dụng hệ thống Intel [I11] để biết thêm chi tiết). Các hệ thống phần cứng khác sử dụng các quy ước khác nhau, nhưng các khái niệm cơ bản là tương tự nhau trên các nền tảng.

Có một chi tiết quan trọng còn lại trong cuộc thảo luận này: làm thế nào mà cái bẫy biết được mã nào để chạy bên trong HĐH? Rõ ràng, quá trình gọi không thể chỉ định một địa chỉ để chuyển đến (như bạn sẽ làm khi thực hiện một cuộc gọi chuyên nghiệp); làm như vậy sẽ cho phép các chương trình nhảy vào bất kỳ đâu vào hạt nhân, đây rõ ràng là một Ý tưởng Rất Xấu1 . Vì vậy, hạt nhân phải kiểm soát cẩn thận những gì mã thực thi trên một cái bẫy.

Kernel làm như vậy bằng cách thiết lập một bảng bẫy tại thời điểm khởi động. Khi máy khởi động, nó sẽ làm như vậy ở chế độ đặc quyền (nhân) và do đó có thể tự do cấu hình phần cứng máy khi cần thiết. Do đó, một trong những điều đầu tiên mà hệ điều hành làm là cho phần cứng biết mã nào sẽ chạy khi các sự kiện ngoại lệ nhất định xảy ra. Ví dụ, mã nào sẽ chạy khi xảy ra ngắt đĩa cứng, khi xảy ra ngắt bàn phím hoặc khi chương trình thực hiện lệnh gọi hệ thống? Hệ điều hành thông báo cho phần cứng của

[1] Hãy tưởng tượng nhảy vào mã để truy cập một tệp, nhưng chỉ sau khi kiểm tra quyền; trên thực tế, rất có thể một khả năng như vậy sẽ cho phép một lập trình viên thông minh để hạt nhân chạy các chuỗi mã tùy ý [S07]. Nói chung, hãy cố gắng tránh những Ý tưởng Rất Xấu như thế này.

OS @ boot (chế độ hạt nhân) khởi	Phần cứng	
tạo bảng bẫy		
	ghi nhớ địa chỉ của ... syscall handler	
OS @ run (chế độ hạt nhân)	**Phần cứng**	**Chương trình (chế độ người dùng)**
Tạo mục nhập cho danh sách quy trình		
Cấp phát bộ nhớ cho chương trình		
Tải chương trình vào bộ nhớ		
Thiết lập ngăn xếp người dùng với argv		
Điền vào ngăn xếp hạt nhân với reg / PC		
return-from-trap		
	khôi phục regs (từ ngăn xếp hạt nhân)	
	chuyển sang chế độ người dùng	
	dùng, chuyển sang chế độ chính	
		Chạy main ()
		...
		Gọi hệ thống bẫy cuộc gọi vào hệ điều hành
	lưu regs (vào ngăn xếp hạt nhân)	
	chuyển sang chế độ hạt nhân	
	nhảy đến trình xử lý bẫy	
Xử lý bẫy		
Thực hiện công việc trả về từ bẫy của syscall		
	khôi phục regs (từ ngăn xếp hạt nhân)	
	chuyển sang chế độ người dùng	
	nhảy đến máy tính sau khi bẫy	
		...
		quay trở lại từ bẫy chính (thông qua lỗi ra ())
Bộ nhớ miễn phí của quá trình		
Xóa khỏi danh sách quy trình		

Hình 6.2: Giao thức thực thi trực tiếp có giới hạn

vị trí của những bộ xử lý bẫy này, thường có một số loại đặc biệt trong kết cấu. Khi phần cứng được thông báo, nó sẽ ghi nhớ vị trí của các trình xử lý này cho đến khi máy được khởi động lại lần sau và do đó phần cứng biết phải làm gì (tức là mã nào cần chuyển đến) khi các cuộc gọi hệ thống và các sự kiện ngoại lệ khác diễn ra.

MẸO: HÃY CỐ GẮNG ĐẦU VÀO CỦA NGƯỜI DÙNG TRONG HỆ THỐNG BẢO MẬT Mặc dù chúng tôi đã rất nỗ lực để bảo vệ HĐH trong các cuộc gọi hệ thống (bằng cách thêm cơ chế bẫy phần cứng và đảm bảo tất cả các cuộc gọi đến HĐH đều được chuyển qua nó), vẫn còn nhiều điều khác các khía cạnh để đề cập đến một **hệ điều hành** an toàn mà chúng ta phải xem xét. Một trong số đó là việc xử lý các đối số ở ranh giới cuộc gọi hệ thống; hệ điều hành phải kiểm tra những gì người dùng chuyển vào và đảm bảo rằng các đối số được chỉ định chính xác, hoặc từ chối cuộc gọi.

Ví dụ, với một **lệnh gọi hệ thống** write () , người dùng chỉ định một địa chỉ của bộ đệm làm nguồn của lệnh gọi ghi. Nếu người dùng (vô tình hoặc ác ý) chuyển đến một địa chỉ "xấu" (ví dụ: một địa chỉ bên trong phần nhân của không gian địa chỉ), HĐH phải phát hiện ra địa chỉ này và từ chối cuộc gọi.
Nếu không, người dùng có thể đọc tất cả bộ nhớ hạt nhân; cho rằng bộ nhớ kernel (ảo) cũng thường bao gồm tất cả bộ nhớ vật lý của hệ thống, phiếu nhỏ này sẽ cho phép một chương trình đọc bộ nhớ của bất kỳ quá trình nào khác trong hệ thống.

Nói chung, một hệ thống an toàn phải xử lý các đầu vào của người dùng một cách đáng ngờ. Không làm như vậy chắc chắn sẽ dẫn đến phần mềm dễ bị tấn công, cảm giác tuyệt vọng rằng thế giới là một nơi không an toàn và đáng sợ, và mất an toàn công việc đối với nhà phát triển hệ điều hành quá tin tưởng.

Để chỉ định chính xác cuộc gọi hệ thống, một số cuộc gọi hệ thống thường được ký cho mỗi cuộc gọi hệ thống. Do đó, mã người dùng chịu trách nhiệm đặt số cuộc gọi hệ thống mong muốn vào sổ đăng ký hoặc tại một vị trí xác định trên ngăn xếp; Hệ điều hành, khi xử lý lệnh gọi hệ thống bên trong trình xử lý bẫy, kiểm tra số này, đảm bảo nó hợp lệ, và nếu có, thực thi mã tương ứng. Mức độ chuyển hướng này đóng vai trò như một hình thức bảo vệ; mã người dùng không thể chỉ định một địa chỉ chính xác để chuyển đến, mà phải yêu cầu một dịch vụ cụ thể thông qua số.

Một điều cuối cùng sang một bên: có thể thực hiện lệnh để thông báo cho phần cứng biết vị trí của các bảng bẫy là một khả năng rất mạnh mẽ. Vì vậy, như bạn có thể đoán, nó cũng là một **hoạt động** đặc quyền . Nếu bạn cố gắng exe dễ thương hướng dẫn này ở chế độ người dùng, phần cứng sẽ không cho phép bạn và bạn có thể đoán điều gì sẽ xảy ra (gợi ý: adios, chương trình vi phạm).
Điểm đáng suy ngẫm: bạn có thể làm gì khủng khiếp với một hệ thống nếu bạn có thể lắp đặt bảng bẫy của riêng mình? Bạn có thể tiếp quản máy không?

Dòng thời gian (với thời gian tăng dần xuống, trong Hình 6.2) tóm tắt giao thức. Chúng ta giả sử rằng mỗi tiến trình có một ngăn xếp hạt nhân nơi các bộ đếm reg (bao gồm các thanh ghi mục đích chung và bộ đếm chương trình) được lưu vào và khôi phục từ (bởi phần cứng) khi chuyển tiếp vào và ra khỏi hạt nhân.

Có hai giai đoạn trong giao thức thực thi trực tiếp hạn chế (LDE) .
Trong lần đầu tiên (lúc khởi động), hạt nhân khởi tạo bảng bẫy và CPU ghi nhớ vị trí của nó để sử dụng tiếp theo. Kernel làm như vậy thông qua một lệnh đặc quyền (tất cả các lệnh đặc quyền đều được tô đậm).

Trong lần thứ hai (khi chạy một tiến trình), hạt nhân thiết lập một vài thứ (ví dụ: cấp phát một nút trên danh sách tiến trình, cấp phát bộ nhớ) trước khi chúng ta nhập một lệnh trả về từ bẫy để bắt đầu thực thi tiến trình; điều này sẽ chuyển CPU sang chế độ người dùng và bắt đầu chạy quá trình.
Khi tiến trình muốn thực hiện một lệnh gọi hệ thống, nó sẽ mắc kẹt lại vào Hệ điều hành, nơi xử lý nó và một lần nữa trả lại quyền kiểm soát thông qua bẫy trả về cho tiến trình. Quá trình này sau đó hoàn thành công việc của nó và trả về từ main (); điều này thường sẽ trở lại thành một số mã sơ khai sẽ thoát khỏi chương trình một cách chính xác (giả sử, bằng cách gọi lệnh gọi hệ thống exit () , bẫy vào HĐH). Tại thời điểm này, hệ điều hành đã được dọn dẹp và chúng ta đã hoàn tất.

6.3 Vấn đề # 2: Chuyển đổi giữa các quá trình

Vấn đề tiếp theo với thực thi trực tiếp là đạt được sự chuyển đổi giữa các quy trình. Chuyển đổi giữa các quy trình sẽ đơn giản, phải không? Hệ điều hành chỉ nên quyết định dừng một quá trình và bắt đầu một quá trình khác. Vấn đề lớn là gì? Nhưng nó thực sự là một chút phức tạp: cụ thể là, nếu một tiến trình đang chạy trên CPU, điều này theo định nghĩa có nghĩa là hệ điều hành không chạy. Nếu hệ điều hành không chạy, làm thế nào nó có thể làm bất cứ điều gì? (gợi ý: không thể) Mặc dù điều này nghe có vẻ triết lý, nhưng đó là một vấn đề thực sự: rõ ràng là không có cách nào để HĐH thực hiện hành động nếu nó không chạy trên CPU. Vì vậy, chúng ta đi đến mấu chốt của vấn đề.

THE CRUX: CÁCH TÁI CHẾ ĐỘ KIỂM SOÁT CỦA CPU
Làm thế nào hệ điều hành có thể lấy lại quyền kiểm soát CPU để nó có thể chuyển đổi giữa các tiến trình?

Phương pháp tiếp cận hợp tác: Chờ cuộc gọi hệ thống Một cách

tiếp cận mà một số hệ thống đã thực hiện trong quá khứ (ví dụ: các phiên bản đầu tiên của hệ điều hành Macintosh [M11], hoặc hệ thống Xerox Alto cũ [A79]) được gọi là phương pháp hợp tác . Theo phong cách này, Hệ điều hành tin tưởng các quy trình của hệ thống sẽ hoạt động hợp lý. Các quá trình chạy quá lâu được cho là định kỳ ngắt CPU để hệ điều hành có thể quyết định chạy một số tác vụ khác.

Vì vậy, bạn có thể hỏi, làm thế nào một quy trình thân thiện lại từ bỏ CPU trong thế giới không tưởng này? Hầu hết các quy trình, hóa ra, chuyển quyền kiểm soát CPU sang hệ điều hành khá thường xuyên bằng cách thực hiện các lệnh gọi hệ thống, chẳng hạn như để mở một tệp và sau đó đọc nó, hoặc để gửi tin nhắn đến một máy khác hoặc để tạo một quy trình mới . Các hệ thống như thế này thường bao gồm một lệnh gọi hệ thống lợi nhuận rõ ràng , không có tác dụng gì ngoại trừ việc chuyển quyền kiểm soát cho HĐH để nó có thể chạy các quy trình khác.

Các ứng dụng cũng chuyển quyền kiểm soát cho HĐH khi chúng làm điều gì đó bất hợp pháp. Ví dụ: nếu một ứng dụng chia cho 0 hoặc cố gắng truy cập vào bộ nhớ mà nó không thể truy cập, nó sẽ tạo ra một cái bẫy đối với

Hệ điều hành. Sau đó, hệ điều hành sẽ có quyền kiểm soát CPU một lần nữa (và có khả năng chấm dứt quá trình vi phạm).

Do đó, trong một hệ thống lập lịch hợp tác, Hệ điều hành giành lại quyền kiểm soát CPU bằng cách chờ một lệnh gọi hệ thống hoặc một hoạt động bất hợp pháp nào đó diễn ra. Bạn cũng có thể đang nghĩ: không phải cách tiếp cận thụ động này ít hơn lý tưởng sao? Điều gì sẽ xảy ra, chẳng hạn, nếu một quá trình (cho dù độc hại, hay chỉ đầy lỗi) kết thúc trong một vòng lặp vô hạn và không bao giờ thực hiện một cuộc gọi hệ thống? Hệ điều hành có thể làm gì sau đó?

Phương pháp tiếp cận không hợp tác: Hệ điều hành giành quyền kiểm soát Nếu

không có một số trợ giúp bổ sung từ phần cứng, hóa ra hệ điều hành không thể làm được gì nhiều khi một quy trình từ chối thực hiện các lệnh gọi hệ thống (hoặc nhầm lẫn) và do đó trả lại quyền kiểm soát cho hệ điều hành. Trên thực tế, trong cách tiếp cận hợp tác, cách duy nhất của bạn khi một quy trình bị mắc kẹt trong một vòng lặp vô hạn là sử dụng giải pháp lâu đời cho tất cả các vấn đề trong hệ thống máy tính: khởi động lại máy. Do đó, chúng ta lại đến một vấn đề con của nhiệm vụ chung là giành quyền kiểm soát CPU.

THE CRUX: LÀM THẾ NÀO ĐỂ KIỂM SOÁT MÀ KHÔNG CẦN HỢP TÁC Làm thế nào HĐH có thể giành quyền kiểm soát CPU ngay cả khi các tiến trình không hợp tác? Hệ điều hành có thể làm gì để đảm bảo quá trình giả mạo không chiếm lấy máy tính?

Câu trả lời hóa ra rất đơn giản và đã được một số người xây dựng hệ thống máy tính phát hiện ra từ nhiều năm trước: một bộ đếm thời gian ngắt [M + 63]. Một thiết bị hẹn giờ có thể được lập trình để tăng gián đoạn sau mỗi mili giây; khi ngắt được nâng lên, quá trình hiện đang chạy bị tạm dừng và trình xử lý ngắt được cấu hình trước trong HĐH sẽ chạy.
Tại thời điểm này, HĐH đã lấy lại quyền kiểm soát CPU và do đó có thể làm những gì nó muốn: dừng quá trình hiện tại và bắt đầu một quá trình khác.

Như chúng ta đã thảo luận trước với các cuộc gọi hệ thống, HĐH phải thông báo cho phần cứng biết mã nào sẽ chạy khi xảy ra ngắt bộ định thời; do đó, tại thời điểm khởi động, hệ điều hành thực hiện chính xác điều đó. Thứ hai, cũng trong trình tự khởi động, hệ điều hành phải khởi động bộ đếm thời gian, điều này tất nhiên là một đặc quyền

MẸO: GIAO DỊCH VỚI ỨNG DỤNG MISBEHAVIOR
Hệ điều hành thường phải đối phó với các quy trình hoạt động sai, những quy trình thông qua thiết kế (độc hại) hoặc tai nạn (lỗi) cố gắng thực hiện điều gì đó mà chúng không nên làm. Trong các hệ thống hiện đại, cách hệ điều hành cố gắng xử lý các lỗi như vậy là chỉ cần chấm dứt kẻ vi phạm. Một cuộc đình công và bạn ra ngoài! Có lẽ tàn bạo, nhưng hệ điều hành nên làm gì khác khi bạn cố gắng truy cập bộ nhớ bất hợp pháp hoặc thực hiện một lệnh bất hợp pháp?

hoạt động. Khi bộ đếm thời gian đã bắt đầu, hệ điều hành có thể cảm thấy an toàn trong điều khiển đó cuối cùng sẽ được trả lại cho nó và do đó, hệ điều hành có thể tự do chạy các chương trình của người dùng. Bộ đếm thời gian cũng có thể được tắt (cũng là một đặc quyền của opera), điều gì đó chúng ta sẽ thảo luận sau khi chúng ta hiểu về đồng thời một cách chi tiết hơn.

Lưu ý rằng phần cứng có một số trách nhiệm khi một con trỏ gián đoạn, đặc biệt là lưu đủ trạng thái của chương trình đang chạy khi xảy ra ngắt để lệnh bẫy trả về tiếp theo có thể tiếp tục chương trình đang chạy một cách chính xác.
Tập hợp các hành động này khá giống với hành vi của phần cứng trong một bẫy gọi hệ thống rõ ràng vào hạt nhân, với các thanh ghi khác nhau do đó được lưu (ví dụ: vào ngăn xếp hạt nhân) và do đó dễ dàng khôi phục bằng lệnh trả về từ bẫy .

Lưu và khôi phục bối cảnh Bây giờ

hệ điều hành đã lấy lại quyền kiểm soát, cho dù hợp tác thông qua lệnh gọi hệ thống hay mạnh mẽ hơn thông qua ngắt bộ hẹn giờ, bạn phải đưa ra quyết định: tiếp tục chạy quy trình hiện đang chạy hay chuyển sang một quy trình khác một. Quyết định này được thực hiện bởi một phần của hệ điều hành được gọi là bộ lập lịch; chúng ta sẽ thảo luận rất chi tiết về các chính sách lập lịch trình trong vài chương tiếp theo.

Nếu quyết định chuyển đổi được thực hiện, hệ điều hành sau đó sẽ thực thi một đoạn mã cấp thấp mà chúng tôi gọi là chuyển đổi ngữ cảnh. Chuyển đổi ngữ cảnh rất đơn giản về mặt khái niệm: tất cả những gì hệ điều hành phải làm là lưu một vài giá trị đăng ký cho quá trình hiện đang thực thi (ví dụ: vào ngăn xếp hạt nhân của nó) và khôi phục một vài giá trị cho quá trình sắp thực thi (từ ngăn xếp hạt nhân của nó). Bằng cách đó, hệ điều hành đảm bảo rằng khi lệnh trả về từ bẫy cuối cùng được thực thi, thay vì quay trở lại quá trình đang chạy, hệ thống tiếp tục thực hiện một quá trình khác.

Để lưu bối cảnh của quá trình hiện đang chạy, hệ điều hành sẽ xuất hiện một số mã lắp ráp cấp thấp để lưu các mục đích chung regis ters, PC và con trỏ ngăn xếp hạt nhân của quá trình hiện đang chạy, sau đó khôi phục các thanh ghi nói trên, PC và chuyển sang ngăn xếp hạt nhân cho quá trình sắp thực thi. Bằng cách chuyển đổi ngăn xếp, hạt nhân sẽ nhập lệnh gọi đến mã chuyển mạch trong ngữ cảnh của một quá trình (quá trình đã bị gián đoạn) và trả về trong ngữ cảnh của một quá trình khác (quá trình sắp thực thi). Khi hệ điều hành cuối cùng thực hiện lệnh trả về từ bẫy,

MẸO: SỬ DỤNG TIMER INTERRUPT ĐỂ KIỂM SOÁT TÁI TẠO

Việc bổ sung ngắt bộ định thời cung cấp cho HĐH khả năng chạy lại trên CPU ngay cả khi các quá trình hoạt động theo kiểu bất hợp tác. Vì vậy, tính năng phần cứng này rất cần thiết trong việc giúp hệ điều hành duy trì quyền kiểm soát máy.

MẸO: REBOOT LÀ HỮU ÍCH Trước đó, chúng tôi đã lưu ý rằng giải pháp duy nhất cho các vòng lặp vô hạn (và các hành vi tương tự) dưới quyền ưu tiên hợp tác là khởi động lại máy. Trong khi bạn có thể chế giễu vụ hack này, các nhà nghiên cứu đã chỉ ra rằng khởi động lại (hoặc trong gen eral, bắt đầu lại một số phần mềm) có thể là một công cụ cực kỳ hữu ích trong việc xây dựng hệ thống mạnh mẽ [C + 04].
Cụ thể, khởi động lại rất hữu ích vì nó chuyển phần mềm trở lại trạng thái đã biết và có khả năng được kiểm tra nhiều hơn. Khởi động lại cũng lấy lại các nguồn tái cũ hoặc bị rò rỉ (ví dụ: bộ nhớ) mà có thể khó xử lý. Cuối cùng, khởi động lại dễ dàng tự động hóa. Vì tất cả những lý do này, không có gì lạ trong các dịch vụ Internet cụm quy mô lớn cho phép phần mềm quản lý hệ thống khởi động lại định kỳ các bộ máy nhằm thiết lập lại chúng và do đó có được những lợi ích được liệt kê ở trên.

Vì vậy, lần sau khi bạn khởi động lại, bạn không chỉ thực hiện một số vụ hack xấu xí. Thay vào đó, bạn đang sử dụng phương pháp đã được kiểm tra theo thời gian để cải thiện hoạt động của hệ thống máy tính. Làm tốt!

quá trình sắp thực thi trở thành quá trình hiện đang chạy.
Và như vậy quá trình chuyển đổi ngữ cảnh đã hoàn thành.

Lịch trình của toàn bộ quá trình được thể hiện trong Hình 6.3. Trong ví dụ này, Quy trình A đang chạy và sau đó bị ngắt bởi bộ định thời gian ngắt. Phần cứng lưu các thanh ghi của nó (vào ngăn xếp nhân của nó) và đi vào nhân (chuyển sang chế độ nhân). Trong trình xử lý ngắt bộ định thời, HĐH quyết định chuyển từ đang chạy Quy trình A sang Quy trình B. Tại thời điểm đó, nó gọi quy trình switch () , cẩn thận lưu các giá trị thanh ghi hiện tại (vào cấu trúc quy trình của A), khôi phục các thanh ghi của Quy trình B (từ mục nhập cấu trúc quy trình của nó), và sau đó chuyển đổi ngữ cảnh, cụ thể là bằng cách thay đổi con trỏ ngăn xếp để sử dụng ngăn xếp hạt nhân của B (chứ không phải của A). Cuối cùng, hệ điều hành trả về từ bẫy, khôi phục các thanh ghi của B và bắt đầu chạy nó.

Lưu ý rằng có hai kiểu lưu / khôi phục thanh ghi xảy ra trong giao thức này. Đầu tiên là khi ngắt bộ định thời xảy ra; trong trường hợp này, các đăng ký người dùng của tiến trình đang chạy được phần cứng lưu ngầm, sử dụng ngăn xếp hạt nhân của tiến trình đó. Thứ hai là khi HĐH quyết định chuyển từ A sang B; trong trường hợp này, các thanh ghi hạt nhân được phần mềm (tức là HĐH) lưu một cách rõ ràng, nhưng lần này vào bộ nhớ trong cấu trúc tiến trình của tiến trình. Hành động thứ hai chuyển hệ thống đang chạy như thể nó vừa bị mắc kẹt vào hạt nhân từ A sang như thể nó vừa bị mắc kẹt vào hạt nhân từ B.

Để bạn hiểu rõ hơn về cách một chuyển đổi như vậy được thực hiện, Hình 6.4 cho thấy mã chuyển đổi ngữ cảnh cho xv6. Xem liệu bạn có thể hiểu được nó không (bạn sẽ phải biết một chút về x86, cũng như một số xv6, để làm như vậy). Các cấu trúc ngữ cảnh cũ và mới lần lượt được tìm thấy trong cấu trúc quy trình của quy trình cũ và mới.

OS @ boot (chế độ hạt nhân) khởi	Phần cứng	
tạo bảng bẫy		
	ghi nhớ địa chỉ của ... trình xử lý hẹn giờ syscall handler	
bắt đầu hẹn giờ ngắt		
	khởi động bộ đếm thời gian ngắt CPU trong X mili giây	
		Chương trình (chế độ người dùng)
		Quy trình A ...
	bộ đếm thời gian ngắt lưu regs (A) k-stack (A) di chuyển đến chế độ hạt nhân nhảy đến trình xử lý bẫy	
Xử lý bẫy Call switch () quy trình lưu regs (A) proc t (A) khôi phục regs (B) proc t (B) chuyển sang k-stack (B) return-from-trap (thành B)		
	khôi phục regs (B) k-stack (B) chuyển sang chế độ người dùng chuyển đến PC của B	
		Quy trình B ...

Hình 6.3: Giao thức thực thi trực tiếp có giới hạn (Ngắt bộ định thời)

6.4 Lo lắng về đồng tiền?

Một số bạn, với tư cách là những độc giả chú ý và hay suy nghĩ, có thể bây giờ nghĩ rằng: "Hừm. điều gì sẽ xảy ra khi, trong một cuộc gọi hệ thống, một bộ đếm thời gian ngắt xảy ra?" hoặc "Điều gì xảy ra khi bạn đang xử lý một ngắt và một ngắt khác xảy ra? Điều đó có khó xử lý trong nhân không? " Câu hỏi hay - chúng tôi thực sự có một số hy vọng cho bạn!

Câu trả lời là có, hệ điều hành thực sự cần phải quan tâm đến điều gì sẽ xảy ra nếu, trong quá trình xử lý ngắt hoặc bẫy, một ngắt khác xảy ra.
Trên thực tế, đây là chủ đề chính xác của toàn bộ phần thứ hai của cuốn sách này, về sự đồng thời; chúng tôi sẽ trì hoãn một cuộc thảo luận chi tiết cho đến lúc đó.

Để kích thích sự thèm ăn của bạn, chúng tôi sẽ chỉ phác thảo một số điều cơ bản về cách hệ điều hành xử lý những tình huống khó khăn này. Một điều đơn giản mà hệ điều hành có thể làm là có thể ngắt trong quá trình xử lý ngắt; làm như vậy đảm bảo rằng khi

```
1 # void swtch (struct context ** cũ, struct context * mới);
2 #
3 # Lưu ngữ cảnh đăng ký hiện tại trong cũ
4 # và sau đó tải ngữ cảnh thanh ghi từ mới.
5. Globl swtch
6 mũi:
7       # Lưu sổ đăng ký cũ
..      movl 4 (% esp),% eax # đưa ptr cũ vào eax
9       popl 0 (% eax)                # lưu IP cũ
10      movl% esp, 4 (% eax) # và ngăn xếp
11      movl% ebx, 8 (% eax) # và các đăng ký khác
12      movl% ecx, 12 (% eax)
13      movl% edx, 16 (% eax)
14      movl% esi, 20 (% eax)
15      movl% edi, 24 (% eax)
16      movl% ebp, 28 (% eax)
17
18      # Tải đăng ký mới
19      movl 4 (% esp),% eax # đưa ptr mới vào eax
20      movl 28 (% eax),% ebp # khôi phục các đăng ký khác
21      movl 24 (% eax),% edi
22      movl 20 (% eax),% esi
23      movl 16 (% eax),% edx
24      tháng 12 (% eax),% ecx
25      movl 8 (% eax),% ebx
26      movl 4 (% eax),% esp # stack được chuyển sang đây
27      pushl 0 (% eax) # return addr được đưa vào vị trí
28      ret # cuối cùng trở lại ctxt mới
```

Hình 6.4: Mã chuyển đổi ngữ cảnh xv6

một ngắt đang được xử lý, không có ngắt nào khác sẽ được chuyển đến CPU. Tất nhiên, hệ điều hành phải cẩn thận khi làm như vậy; vô hiệu hóa gián đoạn cho quá lâu có thể dẫn đến mất ngắt, điều này không tốt (về mặt kỹ thuật).

Hệ điều hành cũng đã phát triển một số khóa các lược đồ để bảo vệ quyền truy cập đồng thời vào cấu trúc dữ liệu nội bộ. Điều này cho phép nhiều hoạt động đang diễn ra bên trong hạt nhân tại đồng thời, đặc biệt hữu ích trên các bộ đa xử lý. Như chúng ta sẽ thấy trong Tuy nhiên, phần tiếp theo của cuốn sách này về tính đồng thời, việc khóa như vậy có thể được thực hiện và dẫn đến nhiều lỗi thú vị và khó tìm.

6.5 Tóm tắt

Chúng tôi đã mô tả một số cơ chế cấp thấp quan trọng để triển khai CPU ảo hóa, một tập hợp các kỹ thuật mà chúng tôi gọi chung là hạn chế thực hiện trực tiếp. Ý tưởng cơ bản rất đơn giản: chỉ cần chạy chương trình bạn muốn chạy trên CPU, nhưng trước tiên hãy đảm bảo thiết lập phần cứng để hạn chế những gì quá trình có thể thực hiện mà không có sự hỗ trợ của hệ điều hành.

BÊN NGOÀI: CHUYỂN ĐỔI TIẾP THEO BAO LÂU Một câu hỏi tự nhiên mà bạn có thể có là: một thứ như chuyển đổi ngữ cảnh mất bao lâu? Hoặc thậm chí là một cuộc gọi hệ thống? Đối với những người bạn không thích, có một công cụ gọi là lmbench [MS96] đo lường chính xác những điều đó, cũng như một số thước đo hiệu suất khác có thể phù hợp.

Kết quả đã được cải thiện khá nhiều theo thời gian, gần như theo dõi hiệu suất của bộ xử lý. Ví dụ, vào năm 1996 chạy Linux 1.3.37 trên CPU P6 200 MHz, các lệnh gọi hệ thống mất khoảng 4 micro giây và chuyển đổi ngữ cảnh khoảng 6 micro giây [MS96]. Các hệ thống hiện đại hoạt động tốt hơn gần như một hoặc trăm độ lớn, với kết quả dưới micro giây trên các hệ thống có bộ xử lý 2 hoặc 3 GHz.

Cần lưu ý rằng không phải tất cả các hành động của hệ điều hành đều theo dõi CPU trên mỗi hình thức. Theo quan sát của Ousterhout, nhiều hoạt động của HĐH tốn nhiều bộ nhớ và băng thông bộ nhớ không được cải thiện đáng kể như tốc độ bộ xử lý theo thời gian [O90]. Do đó, tùy thuộc vào khối lượng công việc của bạn, việc mua bộ xử lý mới nhất và tốt nhất có thể không tăng tốc hệ điều hành của bạn nhiều như bạn có thể hy vọng.

Cách tiếp cận chung này cũng được thực hiện trong cuộc sống thực. Ví dụ, những bạn có con, hoặc ít nhất, đã nghe nói đến trẻ em, có thể quen thuộc với khái niệm phòng chống trẻ em : khóa tủ, cất giữ những thứ nguy hiểm và đậy ổ cắm điện. Khi phòng đã sẵn sàng, bạn có thể để bé tự do đi lại, đảm bảo an toàn khi biết rằng các khía cạnh nguy hiểm nhất trong phòng đã được hạn chế.

Theo cách tương tự, hệ điều hành "chứng minh" CPU, trước tiên (thời gian khởi động trước) thiết lập các trình xử lý bẫy và khởi động bộ đếm thời gian ngắt, sau đó chỉ chạy các quy trình ở chế độ hạn chế. Bằng cách đó, hệ điều hành có thể cảm thấy khá yên tâm rằng các quy trình có thể chạy hiệu quả, chỉ yêu cầu sự can thiệp của hệ điều hành để thực hiện các hoạt động đặc quyền hoặc khi chúng đã độc quyền CPU quá lâu và do đó cần phải được chuyển ra ngoài.

Do đó, chúng tôi có các cơ chế cơ bản để ảo hóa CPU tại chỗ. Nhưng một câu hỏi lớn vẫn chưa được trả lời: chúng ta nên chạy quá trình nào tại một thời điểm nhất định? Đó là câu hỏi mà người lập lịch trình phải trả lời, và do đó, chủ đề tiếp theo của nghiên cứu của chúng tôi.

BÊN NGOÀI: ĐIỀU KHOẢN VIRTUALIZATION CHÍNH CỦA CPU (CƠ CHẾ) • CPU phải hỗ trợ ít nhất hai chế độ thực thi: chế độ người dùng **được nghiêm ngặt lại và chế độ hạt nhân** đặc quyền (không hạn chế) . • Các ứng dụng người dùng điển hình chạy ở chế độ người dùng và sử dụng lệnh gọi hệ thống để đưa vào hạt nhân để yêu cầu các dịch vụ hệ điều hành.

• Lệnh bẫy lưu cẩn thận trạng thái thanh ghi, thay đổi trạng thái kho cứng sang chế độ hạt nhân, và chuyển vào HĐH đến đích được chỉ định trước: bảng bẫy.

• Khi HĐH hoàn thành việc phục vụ một lệnh gọi hệ thống, nó sẽ quay trở lại chương trình người dùng thông qua một **lệnh đặc biệt khác** từ bẫy , lệnh này cấp lại đặc quyền và trả lại quyền kiểm soát lệnh sau khi bẫy đã xâm nhập vào HĐH.

• Các bảng bẫy phải được HĐH thiết lập tại thời điểm khởi động và đảm bảo rằng chúng không thể được sửa đổi dễ dàng bởi các chương trình người dùng. Tất cả những điều này là một phần của **giao thức** thực thi trực tiếp hạn chế chạy các chương trình một cách hiệu quả nhưng không mất quyền kiểm soát hệ điều hành.

• Khi một chương trình đang chạy, HĐH phải sử dụng các cơ chế phần cứng để đảm bảo chương trình người dùng không chạy mãi mãi, cụ thể là ngắt bộ định thời. Cách tiếp cận này là một cách tiếp cận không hợp tác để lập lịch CPU.

• Đôi khi HĐH, trong thời gian ngắt hẹn giờ hoặc cuộc gọi hệ thống, có thể muốn chuyển từ đang chạy quy trình hiện tại sang một quy trình khác, một kỹ thuật cấp thấp được gọi là chuyển đổi ngữ cảnh.

.

Người giới thiệu

[A79] “Sổ tay người dùng Alto” của Xerox. Trung tâm Nghiên cứu Xerox Palo Alto, tháng 9 năm 1979.
Có sẵn: http://history-computer.com/Library/AltoUsersHandbook.pdf. Một hệ thống tuyệt vời, đi trước thời đại.
Trở nên nổi tiếng vì Steve Jobs đã đến thăm, ghi chép, xây dựng Lisa và cuối cùng là Mac.

[C + 04] “Microreboot - Một kỹ thuật phục hồi giá rẻ” của G. Candea, S. Kawamoto, Y.
Fujiki, G. Friedman, A. Fox. OSDI '04, San Francisco, CA, tháng 12 năm 2004. Một bài báo xuất sắc chỉ ra rằng người ta có thể tiến xa như thế nào với việc khởi động lại trong việc xây dựng các hệ thống mạnh mẽ hơn.

[I11] “Sổ tay dành cho nhà phát triển phần mềm kiến trúc Intel 64 và IA-32” trong Tập 3A và 3B: Hướng dẫn lập trình hệ thống. Tập đoàn Intel, tháng 1 năm 2011. Đây chỉ là một hướng dẫn nhàm chán, nhưng đôi khi chúng rất hữu ích.

[K + 61] “Hệ thống lưu trữ một cấp” của T. Kilburn, DBG Edwards, MJ Lanigan, FH Sumner.
Giao dịch IRE trên Máy tính Điện tử, tháng 4 năm 1962. Atlas đi tiên phong trong phần lớn những gì bạn thấy trong các hệ thống hiện đại. Tuy nhiên, bài báo này không phải là tốt nhất để đọc. Nếu bạn chỉ đọc một cuốn, bạn có thể thử quan điểm lịch sử bên dưới [L78].

[L78] “The Manchester Mark I và Atlas: Một viễn cảnh lịch sử” của SH Lavington. Thông báo về ACM, 21: 1, tháng 1 năm 1978. Lịch sử về sự phát triển ban đầu của máy tính và những nỗ lực tiên phong của Atlas.

[M + 63] “Hệ thống gỡ lỗi chia sẻ thời gian cho máy tính nhỏ” của J. McCarthy, S. Boilen, E. Fredkin, JCR Licklider. AFIPS '63 (Mùa xuân), tháng 5 năm 1963, New York, Hoa Kỳ. Một bài báo ban đầu về chia sẻ thời gian đề cập đến việc sử dụng ngắt bộ đếm thời gian; trích dẫn thảo luận về nó: "Nhiệm vụ cơ bản của quy trình đồng hồ kênh 17 là quyết định xem có nên xóa người dùng hiện tại khỏi lõi hay không và nếu có để quyết định chương trình người dùng nào sẽ hoán đổi khi anh ta ra ngoài."

[MS96] “lmbench: Các công cụ di động để phân tích hiệu suất” của Larry McVoy và Carl Staelin.
Hội nghị kỹ thuật thường niên USENIX, tháng 1 năm 1996. Một bài báo thú vị về cách đo lường một số điều khác nhau về hệ điều hành của bạn và hiệu suất của nó. Tải xuống lmbench và dùng thử.

[M11] “Mac OS 9” của Apple Computer, Inc .. Tháng 1 năm 2011. http://en.wikipedia.org/wiki/ Mac OS 9 . Bạn thậm chí có thể tìm thấy một trình giả lập OS 9 ở đó nếu bạn muốn; kiểm tra nó ra, đó là một Mac nhỏ thú vị!

[O90] “Tại sao Hệ điều hành không hoạt động nhanh hơn như phần cứng?” của J. Ouster hout. Hội nghị mùa hè USENIX, tháng 6 năm 1990. Một bài báo kinh điển về bản chất của hiệu suất hệ điều hành.

[P10] “Đặc tả UNIX Duy nhất, Phiên bản 3” của The Open Group, tháng 5 năm 2010. Sẵn có: http://www.unix.org/version3/. Điều này thật khó và khó đọc, vì vậy có thể tránh nó nếu bạn có thể. Giống như, trừ khi ai đó trả tiền cho bạn để đọc nó. Hoặc, bạn chỉ tò mò đến mức không thể không làm điều đó!

[S07] “Hình học của thịt vô tội trên xương: Return-into-libc không có hàm gọi (trên x86)” của Hovav Shacham. CCS '07, tháng 10 năm 2007. Một trong những ý tưởng tuyệt vời, đầy sáng tạo mà thỉnh thoảng bạn sẽ thấy trong nghiên cứu. Tác giả cho thấy rằng nếu bạn có thể nhảy vào mã tùy ý, về cơ bản bạn có thể ghép nối bất kỳ chuỗi mã nào bạn thích (với một cơ sở mã lớn); đọc bài báo để biết chi tiết. Kỹ thuật này khiến việc phòng thủ trước các cuộc tấn công ác ý thậm chí còn khó hơn.

Bài tập về nhà (Đo lường)

BÊN TRONG: QUAN ĐIỂM VỀ ĐO LƯỜNG Bài tập về nhà đo lường là những bài tập nhỏ trong đó bạn viết mã để chạy trên một máy thực, nhằm đo lường một số khía cạnh của hệ điều hành hoặc hiệu suất phần cứng. Ý tưởng đằng sau các bài tập về nhà như vậy là cung cấp cho bạn một chút kinh nghiệm thực tế với hệ điều hành thực tế.

Trong bài tập về nhà này, bạn sẽ đo lường chi phí của một cuộc gọi hệ thống và chuyển đổi ngữ cảnh. Đo lường chi phí của một cuộc gọi hệ thống là tương đối dễ dàng. Ví dụ, bạn có thể gọi liên tục một lệnh gọi hệ thống đơn giản (ví dụ: thực hiện đọc 0 byte) và thời gian mất bao lâu; chia thời gian cho số lần lặp lại sẽ cho bạn ước tính chi phí của một cuộc gọi hệ thống.

Một điều bạn sẽ phải tính đến là độ chính xác và độ chính xác của bộ đếm thời gian của bạn. Một bộ đếm thời gian điển hình mà bạn có thể sử dụng là gettimeofday (); đọc trang người đàn ông để biết chi tiết. Những gì bạn sẽ thấy ở đó là gettimeofday () trả về thời gian tính bằng micro giây kể từ năm 1970; tuy nhiên, điều này không có nghĩa là bộ đếm thời gian chính xác đến từng micro giây. Đo lường các cuộc gọi liên tiếp đến gettimeofday () để tìm hiểu điều gì đó về mức độ chính xác của đồng minh hẹn giờ; điều này sẽ cho bạn biết bạn sẽ phải chạy bao nhiêu lần lặp lại kiểm tra cuộc gọi hệ thống rỗng của mình để có được kết quả đo lường tốt. Nếu gettimeofday () không đủ chính xác cho bạn, bạn có thể xem xét sử dụng lệnh rdtsc có sẵn trên máy x86.

Đo lường chi phí của một công tắc ngữ cảnh phức tạp hơn một chút. Điểm chuẩn lmbench làm như vậy bằng cách chạy hai quy trình trên một CPU và thiết lập hai ống UNIX giữa chúng; một đường ống chỉ là một trong nhiều cách mà các quá trình trong hệ thống UNIX có thể giao tiếp với nhau. Quá trình đầu tiên sau đó đưa ra một bản ghi vào đường ống đầu tiên và chờ đọc vào lần thứ hai; khi thấy quy trình đầu tiên đang đợi một thứ gì đó đọc từ đường ống thứ hai, Hệ điều hành đặt quá trình đầu tiên ở trạng thái bị chặn và chuyển sang quá trình khác, quá trình này đọc từ đường ống đầu tiên và sau đó ghi vào đường dẫn thứ hai. Khi quá trình thứ hai cố gắng đọc lại từ đường ống đầu tiên, nó sẽ chặn và do đó chu trình liên lạc qua lại tiếp tục. Bằng cách đo lường chi phí giao tiếp như vậy lặp đi lặp lại, lmbench có thể ước tính tốt chi phí của một bộ chuyển đổi ngữ cảnh. Bạn có thể thử tạo lại một cái gì đó tương tự ở đây, bằng cách sử dụng đường ống hoặc có thể là một số cơ chế giao tiếp khác như ổ cắm UNIX .

Một khó khăn trong việc đo lường chi phí chuyển đổi ngữ cảnh phát sinh trong các hệ thống có nhiều hơn một CPU; những gì bạn cần làm trên một hệ thống như vậy là đảm bảo rằng các quy trình chuyển đổi ngữ cảnh của bạn được đặt trên cùng một bộ xử lý. Mặt khác, hầu hết các hệ điều hành đều có các lệnh gọi để liên kết một tiến trình với một bộ xử lý cụ thể; trên Linux, chẳng hạn, lệnh gọi setaffinity () lên lịch là thứ bạn đang tìm kiếm. Bằng cách đảm bảo cả hai quy trình đều trên cùng một bộ xử lý, bạn đảm bảo đo lường chi phí của việc Hệ điều hành dừng một quy trình và khôi phục một quy trình khác trên cùng một CPU.