15

Cơ chế: Dịch địa chỉ

Khi phát triển ảo hóa CPU, chúng tôi tập trung vào một cơ chế chung được gọi là thực thi trực tiếp hạn chế (hoặc LDE). Ý tưởng trở thành LDE rất đơn giản: đối với hầu hết các phần, hãy để chương trình chạy trực tiếp trên phần cứng; tuy nhiên, tại một số thời điểm quan trọng nhất định (chẳng hạn như khi tiến trình đưa ra lệnh gọi hệ thống hoặc xảy ra ngắt bộ hẹn giờ), hãy sắp xếp sao cho HĐH tham gia và đảm bảo điều "đúng đắn" xảy ra. Do đó, hệ điều hành, với một chút hỗ trợ phần cứng, cố gắng hết sức để thoát khỏi chương trình đang chạy, để cung cấp một ảo hóa hiệu quả; tuy nhiên, bằng cách liên tục đặt ra những thời điểm quan trọng đó, Hệ điều hành đảm bảo rằng nó duy trì quyền kiểm soát đối với phần cứng. Hiệu quả và kiểm soát cùng nhau là hai trong số các mục tiêu chính của bất kỳ hệ điều hành hiện đại nào.

Trong ảo hóa bộ nhớ, chúng tôi sẽ theo đuổi một chiến lược tương tự, đạt được cả hiệu quả và khả năng kiểm soát trong khi cung cấp ảo hóa mong muốn. Ef ficiency ra lệnh rằng chúng tôi sử dụng hỗ trợ phần cứng, lúc đầu sẽ khá thô sơ (ví dụ: chỉ một vài thanh ghi) nhưng sẽ phát triển trở nên khá phức tạp (ví dụ: TLB, hỗ trợ bảng trang, v.v., như bạn sẽ làm hiểu). Kiểm soát ngụ ý rằng Hệ điều hành đảm bảo rằng không có ứng dụng nào được phép truy cập vào bất kỳ bộ nhớ nào ngoài bộ nhớ của chính nó; do đó, để bảo vệ các ứng dụng khỏi nhau và hệ điều hành khỏi các ứng dụng, chúng tôi cũng sẽ cần sự trợ giúp từ phần cứng ở đây. Cuối cùng, chúng ta sẽ cần thêm một chút từ hệ thống VM, về tính linh hoạt; đặc biệt, chúng tôi muốn các chương trình có thể sử dụng không gian địa chỉ của chúng theo bất kỳ cách nào chúng muốn, do đó làm cho hệ thống dễ dàng lập trình hơn. Và do đó, chúng tôi đi đến điểm mấu chốt tinh tế:

THE CRUX: LÀM
THẾ NÀO ĐỂ VIRTUALIZE BỘ NHỚ HIỆU QUẢ VÀ LINH HOẠT Làm thế nào chúng ta

có thể xây dựng một ảo hóa bộ nhớ hiệu quả? Làm cách nào để chúng tôi cung cấp sự linh hoạt cần thiết cho các ứng dụng? Làm cách nào để chúng tôi duy trì quyền kiểm soát vị trí bộ nhớ mà ứng dụng có thể truy cập và do đó đảm bảo rằng các quyền truy cập vào bộ nhớ của ứng dụng được giới hạn đúng cách? Làm thế nào để chúng tôi thực hiện tất cả những điều này một cách hiệu quả?

Kỹ thuật chung mà chúng tôi sẽ sử dụng, mà bạn có thể coi là một bổ sung đối với cách tiếp cận chung của chúng tôi về việc thực thi trực tiếp hạn chế, là một cái gì đó được gọi là dịch địa chỉ dựa trên phần cứng, hay chỉ viết tắt là chuyển địa chỉ . Với tính năng dịch địa chỉ, phần cứng sẽ biến đổi từng truy cập bộ nhớ (ví dụ: tìm nạp, tải hoặc lưu trữ lệnh), thay đổi địa chỉ tual nguyên vẹn được cung cấp bởi lệnh thành địa chỉ thực trong đó thông tin mong muốn thực sự được định vị. Vì vậy, trên mỗi và mọi ký ức tham chiếu, một bản dịch địa chỉ được thực hiện bởi phần cứng để chuyển hướng các tham chiếu bộ nhớ ứng dụng đến các vị trí thực tế của chúng trong bộ nhớ.

Tất nhiên, một mình phần cứng không thể ảo hóa bộ nhớ, vì nó chỉ cung cấp cơ chế cấp thấp để làm điều đó một cách hiệu quả. Hệ điều hành phải nhận được tham gia vào các điểm chính để thiết lập phần cứng sao cho diễn ra các hoạt động chuyển đổi chính xác; do đó nó phải quản lý bộ nhớ, theo dõi các địa điểm miễn phí và đang được sử dụng, đồng thời can thiệp một cách thận trọng để duy trì kiểm soát cách bộ nhớ được sử dụng.

Một lần nữa, mục tiêu của tất cả công việc này là tạo ra một Illu sion đẹp: chương trình có bộ nhớ riêng của nó, nơi có mã riêng của nó và dữ liệu cư trú. Đằng sau thực tế ảo đó là sự thật vật lý xấu xí: rằng nhiều chương trình thực sự đang chia sẻ bộ nhớ cùng một lúc, như CPU (hoặc các CPU) chuyển đổi giữa chạy chương trình này và chương trình tiếp theo. Thông qua ảo hóa, hệ điều hành (với sự trợ giúp của phần cứng) biến thực tế máy móc thành một thứ trừu tượng hữu ích, mạnh mẽ và dễ sử dụng.

15.1 Giả định

Những nỗ lực đầu tiên của chúng tôi trong việc ảo hóa bộ nhớ sẽ rất đơn giản, gần như buồn cười như vậy. Hãy tiếp tục, cười tất cả những gì bạn muốn; không lâu nữa nó sẽ là hệ điều hành cười nhạo bạn, khi bạn cố gắng hiểu thông tin chi tiết về TLB, bảng trang nhiều cấp, và các kỳ quan kỹ thuật khác. Không thích ý tưởng của hệ điều hành đang cười nhạo bạn? Chà, bạn có thể gặp may rồi; đó chỉ là cách hệ điều hành hoạt động.

Cụ thể, bây giờ chúng tôi sẽ giả định rằng không gian địa chỉ của người dùng phải được đặt liền kề trong bộ nhớ vật lý. Chúng tôi cũng sẽ giả định rằng, đối với sim plicity, kích thước của không gian địa chỉ không quá lớn; cụ thể là nó nhỏ hơn kích thước của bộ nhớ vật lý. Cuối cùng, chúng tôi cũng sẽ giả định rằng mỗi không gian địa chỉ có cùng kích thước. Đừng lo lắng nếu những giả định này nghe có vẻ không thực tế; chúng tôi sẽ thư giãn chúng khi chúng tôi đi, do đó đạt được ảo hóa thực tế của bộ nhớ.

15.2 Một ví dụ

Để hiểu rõ hơn những gì chúng ta cần làm để triển khai phân phối địa chỉ và tại sao chúng ta cần một cơ chế như vậy, chúng ta hãy xem xét một bài kiểm tra đơn giản. Hãy tưởng tượng có một quá trình có không gian địa chỉ như được chỉ ra trong Hình 15.1. Những gì chúng ta sẽ kiểm tra ở đây là một chuỗi mã ngắn tải một giá trị từ bộ nhớ, tăng nó lên ba và sau đó lưu trữ giá trị trở lại bộ nhớ. Bạn có thể tưởng tượng trình bày lại bằng ngôn ngữ C của đoạn mã này có thể trông như thế này:

MẸO: LIÊN KẾT LÀ MẠNH MẼ Sự xen kẽ

là một kỹ thuật chung chung và mạnh mẽ thường được sử dụng để tạo ra hiệu quả lớn trong các hệ thống máy tính. Trong bộ nhớ ảo hóa, phần cứng sẽ thực hiện mỗi lần truy cập bộ nhớ và dịch từng địa chỉ ảo do tiến trình cấp thành địa chỉ vật lý nơi thực sự lưu trữ thông tin mong muốn. Tuy nhiên, kỹ thuật xen kẽ chung được áp dụng rộng rãi hơn nhiều; thực sự, hầu hết mọi giao diện được xác định rõ đều có thể được xen kẽ vào nhau, để thêm chức năng mới hoặc cải thiện một số khía cạnh khác của hệ thống. Một trong những lợi ích thông thường của cách tiếp cận như vậy là tính minh bạch; sự xen kẽ thường được thực hiện mà không làm thay đổi bộ mặt bên trong của khách hàng, do đó không yêu cầu thay đổi đối với khách hàng đã nói.

```
void func () {
     int x = 3000; // cảm ơn, Perry. x = x +
     3; // dòng mã mà chúng tôi quan tâm
     ...
```

Trình biên dịch biến dòng mã này thành assembly, có thể trông giống như thế này (trong x86 assembly). Sử dụng objdump trên Linux hoặc otool trên Mac để tháo rời nó:

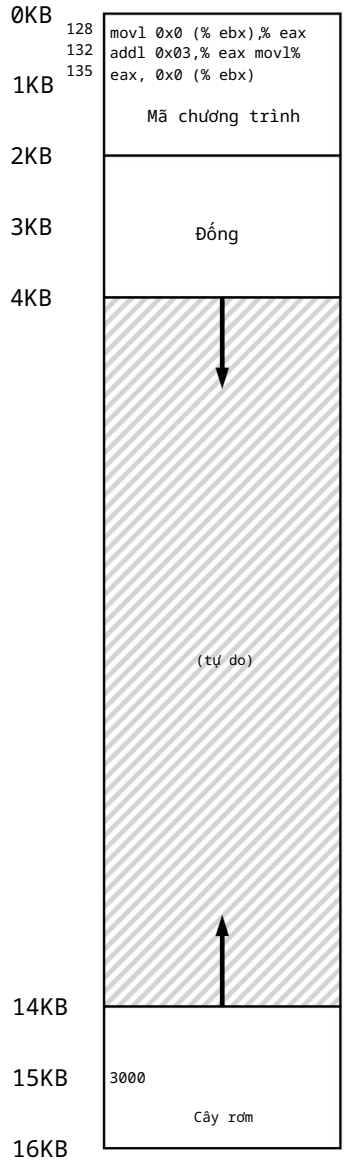
```
128: movl 0x0 (% ebx),% eax 132:        ; tải 0 + ebx vào eax;
addl $ 0x03,% eax 135: movl%            thêm 3 vào thanh ghi eax;
eax, 0x0 (% ebx)                        lưu trữ eax trở lại mem
```

Đoạn mã này tương đối đơn giản; nó giả định rằng địa chỉ của x đã được đặt trong thanh ghi ebx, và sau đó tải giá trị tại địa chỉ đó vào thanh ghi mục đích chung eax bằng cách sử dụng movl trong struction (đối với di chuyển "longword"). Lệnh tiếp theo thêm 3 vào eax, và lệnh cuối cùng lưu giá trị trong eax trở lại bộ nhớ tại cùng vị trí đó.

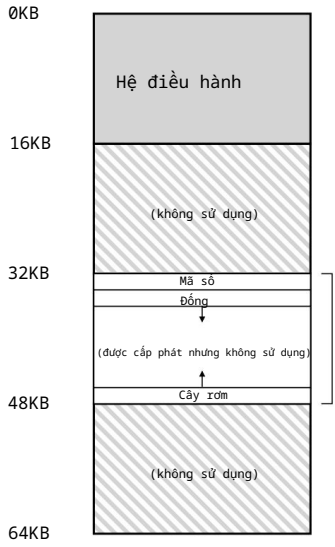
Trong Hình 15.1 (trang 4), hãy quan sát cách cả mã và dữ liệu được trình bày trong không gian địa chỉ của quy trình; chuỗi mã ba lệnh nằm ở địa chỉ 128 (trong phần mã gần trên cùng) và giá trị của biến x ở địa chỉ 15 KB (trong ngăn xếp gần dưới cùng). Trong hình, giá trị ban đầu của x là 3000, như thể hiện ở vị trí của nó trên ngăn xếp.

Khi các hướng dẫn này chạy, từ quan điểm của quá trình, các truy cập bộ nhớ sau đây sẽ diễn ra.

• Tìm nạp lệnh tại địa chỉ 128 • Thực hiện lệnh này (tải từ địa chỉ 15 KB) • Tìm nạp lệnh tại địa chỉ 132 • Thực hiện lệnh này (không có tham chiếu bộ nhớ) • Tìm nạp lệnh tại địa chỉ 135 • Thực hiện lệnh này (lưu trữ đến địa chỉ 15 KB)



Hình 15.1: Một tiến trình và không gian địa chỉ của nó



Hình 15.2: Bộ nhớ vật lý với một quá trình di dời duy nhất Từ quan điểm

của chương trình, không gian địa chỉ của nó bắt đầu từ địa chỉ 0 và phát triển đến tối đa 16 KB; tất cả các tham chiếu bộ nhớ mà nó tạo ra phải nằm trong các giới hạn này. Tuy nhiên, để ảo hóa bộ nhớ, HĐH muốn đặt tiến trình ở một nơi khác trong bộ nhớ vật lý, không nhất thiết phải ở địa chỉ 0. Vì vậy, chúng ta có vấn đề: làm thế nào chúng ta có thể định vị lại tiến trình này trong bộ nhớ theo cách minh bạch với tiến trình. ? Làm thế nào chúng ta có thể cung cấp ảo tưởng về một không gian địa chỉ ảo bắt đầu từ 0, trong khi trong thực tế, không gian địa chỉ được đặt tại một số địa chỉ vật lý khác?

Ví dụ về bộ nhớ vật lý có thể trông như thế nào khi không gian địa chỉ của pro cess này được đặt vào bộ nhớ được tìm thấy trong Hình 15.2.
Trong hình, bạn có thể thấy HĐH sử dụng khe bộ nhớ vật lý đầu tiên cho chính nó và nó đã chuyển quá trình từ ví dụ trên vào khe bắt đầu ở địa chỉ bộ nhớ vật lý 32 KB. Hai khe còn lại miễn phí (16 KB-32 KB và 48 KB-64 KB).

15.3 Chuyển vị trí động (dựa trên phần cứng) Để có được một

số hiểu biết về dịch địa chỉ dựa trên phần cứng, trước tiên chúng ta sẽ thảo luận về hiện thân đầu tiên của nó. Được giới thiệu trong các máy chia sẻ thời gian đầu tiên vào cuối những năm 1950 là một ý tưởng đơn giản được gọi là cơ sở và giới hạn; kỹ thuật này cũng được gọi là tái định vị động; chúng tôi sẽ sử dụng cả hai thuật ngữ thay thế cho nhau [SS74].

Cụ thể, chúng ta sẽ cần hai thanh ghi phần cứng trong mỗi CPU: một được gọi là thanh ghi cơ sở và thanh ghi còn lại là giới hạn (đôi khi được gọi là đăng ký giới hạn). Cặp cơ sở và giới hạn này sẽ cho phép chúng tôi đặt

BÊN NGOÀI: RELOCATION DỰA TRÊN PHẦN MỀM

Trong những ngày đầu, trước khi hỗ trợ phần cứng xuất hiện, một số hệ thống đã hình thành một dạng di dời thô sơ hoàn toàn thông qua các phương pháp phần mềm. Kỹ thuật cơ bản được gọi là tái định vị tĩnh, trong đó một phần mềm được gọi là bộ tải lấy một tệp thực thi sắp được chạy và ghi lại các địa chỉ của nó vào khoảng trống mong muốn trong bộ nhớ vật lý.

Ví dụ: nếu một lệnh là một tải từ địa chỉ 1000 vào một bộ đếm reg (ví dụ: movl 1000,% eax) và không gian địa chỉ của chương trình được tải bắt đầu từ địa chỉ 3000 (chứ không phải 0, như chương trình nghĩ), trình nạp sẽ viết lại lệnh để bù đắp mỗi địa chỉ bằng 3000 (ví dụ: movl 4000,% eax). Bằng cách này, một sự tái định vị tĩnh đơn giản của không gian địa chỉ của tiến trình sẽ đạt được.

Tuy nhiên, việc di dời tĩnh có vô số vấn đề. Đầu tiên và quan trọng nhất, nó không cung cấp sự bảo vệ, vì các tiến trình có thể tạo ra các địa chỉ xấu và do đó truy cập bất hợp pháp vào bộ nhớ của tiến trình khác hoặc thậm chí hệ điều hành; nói chung, có thể cần hỗ trợ phần cứng để bảo vệ thực sự [WL + 93]. Một tiêu cực khác là một khi đã được đặt, sau này rất khó để di dời một không gian trang phục quảng cáo đến một vị trí khác [M65].

không gian địa chỉ ở bất kỳ nơi nào chúng ta muốn trong bộ nhớ vật lý và làm như vậy trong khi đảm bảo rằng quá trình chỉ có thể truy cập không gian địa chỉ của chính nó.

Trong thiết lập này, mỗi chương trình được viết và biên dịch như thể nó được tải ở địa chỉ số không. Tuy nhiên, khi một chương trình bắt đầu chạy, hệ điều hành sẽ quyết định nơi nó sẽ được tải trong bộ nhớ vật lý và đặt thanh ghi cơ sở thành giá trị đó. Trong ví dụ trên, HĐH quyết định tải tiến trình ở địa chỉ vật lý 32 KB và do đó đặt thanh ghi cơ sở thành giá trị này.

Những điều thú vị bắt đầu xảy ra khi tiến trình đang chạy. Bây giờ, khi bất kỳ tham chiếu bộ nhớ nào được tạo bởi quá trình, nó sẽ được bộ xử lý dịch theo cách sau:

địa chỉ thực = địa chỉ ảo + cơ sở

Mỗi tham chiếu bộ nhớ được tạo ra bởi quá trình là một địa chỉ ảo; phần cứng lần lượt thêm nội dung của thanh ghi cơ sở vào địa chỉ này và kết quả là một địa chỉ vật lý có thể được cấp cho hệ thống bộ nhớ.

Để hiểu rõ hơn điều này, chúng ta hãy theo dõi những gì sẽ xảy ra khi một lệnh duy nhất được thực thi. Cụ thể, hãy xem một hướng dẫn từ trình tự trước đó của chúng tôi:

```
128: movl 0x0 (% ebx),% eax
```

Bộ đếm chương trình (PC) được đặt thành 128; khi phần cứng cần tìm nạp lệnh này, trước tiên nó sẽ thêm giá trị vào giá trị thanh ghi cơ sở là 32 KB (32768) để có được địa chỉ vật lý là 32896; phần cứng sau đó tìm nạp lệnh từ địa chỉ vật lý đó. Tiếp theo, bộ xử lý bắt đầu thực hiện lệnh. Tại một số thời điểm, quy trình sau đó sẽ phát hành

LỜI KHUYÊN: LIÊN QUAN ĐẾN ĐỘNG HỌC DỰA TRÊN PHẦN CỨNG

Với việc di dời động, một chút phần cứng sẽ đi được một chặng đường dài. Cụ thể, một thanh ghi cơ sở được sử dụng để biến đổi các địa chỉ ảo (do pro gram tạo ra) thành các địa chỉ vật lý. Một thanh ghi giới hạn (hoặc giới hạn) đảm bảo rằng các địa chỉ đó nằm trong giới hạn của không gian địa chỉ. Chúng cùng nhau cung cấp một ảo hóa bộ nhớ đơn giản và hiệu quả.

tải từ địa chỉ ảo 15 KB, mà bộ xử lý lấy và thêm một lần nữa vào thanh ghi cơ sở (32 KB), nhận được địa chỉ vật lý cuối cùng là 47 KB và do đó có nội dung mong muốn.

Chuyển đổi một địa chỉ ảo thành một địa chỉ vật lý chính xác là kỹ thuật mà chúng tôi gọi là dịch địa chỉ; nghĩa là, phần cứng nhận một địa chỉ ảo mà quá trình nghĩ rằng nó đang tham chiếu và biến nó thành một địa chỉ vật lý, nơi dữ liệu thực sự cư trú. Bởi vì việc di chuyển địa chỉ này xảy ra trong thời gian chạy và bởi vì chúng ta có thể di chuyển các không gian địa chỉ ngay cả sau khi tiến trình đã bắt đầu chạy, kỹ thuật này thường được gọi là tái định vị động [M65].

Bây giờ bạn có thể hỏi: điều gì đã xảy ra với giới hạn đăng ký (giới hạn) đó? Rốt cuộc, đây không phải là cách tiếp cận cơ sở và giới hạn? Quả thực là như vậy. Như bạn có thể đã đoán, thanh ghi giới hạn ở đó để trợ giúp với tion protec. Cụ thể, trước tiên bộ xử lý sẽ kiểm tra xem tham chiếu bộ nhớ có nằm trong giới hạn hay không để đảm bảo rằng nó là hợp pháp; trong ví dụ đơn giản ở trên, thanh ghi giới hạn sẽ luôn được đặt thành 16 KB. Nếu một quá trình tạo ra một địa chỉ tual lớn hơn giới hạn hoặc một địa chỉ âm, CPU sẽ đưa ra một ngoại lệ và quá trình có thể sẽ bị kết thúc.
Do đó, điểm của giới hạn là đảm bảo rằng tất cả các địa chỉ được tạo bởi quy trình đều hợp pháp và nằm trong "giới hạn" của quy trình.

Chúng ta nên lưu ý rằng thanh ghi cơ sở và giới hạn là các dữ liệu cấu trúc phần cứng được lưu giữ trên chip (một cặp cho mỗi CPU). Đôi khi người ta gọi phần của bộ xử lý giúp dịch địa chỉ là đơn vị quản lý bộ nhớ (MMU); khi chúng tôi phát triển các kỹ thuật quản lý bộ nhớ phức tạp hơn, chúng tôi sẽ bổ sung thêm nhiều mạch cho MMU.

Một phần nhỏ về thanh ghi ràng buộc, có thể được định nghĩa theo một trong hai cách. Theo một cách (như trên), nó giữ kích thước của không gian địa chỉ và do đó phần cứng sẽ kiểm tra địa chỉ ảo so với nó trước khi thêm cơ sở. Theo cách thứ hai, nó giữ địa chỉ vật lý của phần cuối của không gian địa chỉ, và do đó, phần cứng trước tiên sẽ thêm cơ sở và sau đó đảm bảo địa chỉ nằm trong giới hạn. Cả hai phương pháp đều tương đương về mặt logic; để đơn giản, chúng tôi thường giả định phương pháp cũ.

Bản dịch ví dụ Để hiểu

bản dịch địa chỉ qua base-and-bounds một cách chi tiết hơn, chúng ta hãy xem một ví dụ. Hãy tưởng tượng một quá trình có không gian địa chỉ có kích thước 4 KB (vâng, nhỏ không thực tế) đã được tải ở địa chỉ vật lý 16 KB. Đây là kết quả của một số bản dịch địa chỉ:

Địa chỉ ảo 0		Địa chỉ vật lý
		16 KB
1 KB		17 KB
3000		19384
4400	Lỗi (ngoài giới hạn)	

Như bạn có thể thấy từ ví dụ, bạn có thể dễ dàng thêm địa chỉ cơ sở đến địa chỉ ảo (có thể được xem như một bù vào không gian địa chỉ) để lấy địa chỉ vật lý kết quả. Chỉ có nếu địa chỉ ảo "quá lớn" hoặc âm thì kết quả là lỗi, khiến một ngoại lệ được nâng lên.

15.4 Hỗ trợ phần cứng: Tóm tắt

Bây giờ chúng ta hãy tóm tắt sự hỗ trợ mà chúng ta cần từ phần cứng (cũng xem Hình 15.3, trang 9). Đầu tiên, như đã thảo luận trong chương về ảo hóa CPU, chúng tôi yêu cầu hai chế độ CPU khác nhau. Hệ điều hành chạy trong đặc quyền chế độ (hoặc chế độ hạt nhân), nơi nó có quyền truy cập vào toàn bộ máy; ứng dụng chạy ở chế độ người dùng, nơi chúng bị hạn chế về những gì chúng có thể làm. Một bit đơn, có lẽ được lưu trữ trong một số loại từ trạng thái bộ xử lý, chỉ ra chế độ CPU hiện đang chạy; khi đặc biệt nhất định
các dịp (ví dụ: một cuộc gọi hệ thống hoặc một số loại ngoại lệ hoặc gián đoạn khác), CPU chuyển đổi chế độ.

Phần cứng cũng phải cung cấp cơ sở và giới hạn đăng ký bản thân của chúng; do đó mỗi CPU có thêm một cặp thanh ghi, một phần của đơn vị quản lý mem ory (MMU) của CPU. Khi chương trình người dùng đang chạy ning, phần cứng sẽ dịch từng địa chỉ, bằng cách thêm giá trị cơ sở
đến địa chỉ ảo do chương trình người dùng tạo ra. Phần cứng phải cũng có thể kiểm tra xem địa chỉ có hợp lệ hay không, đã hoàn thành bằng cách sử dụng thanh ghi giới hạn và một số mạch trong CPU.

Phần cứng phải cung cấp các hướng dẫn đặc biệt để sửa đổi cơ sở và giới hạn các thanh ghi, cho phép HĐH thay đổi chúng khi khác các quy trình chạy. Các hướng dẫn này là đặc quyền; chỉ trong chế độ kernel (hoặc priv ileged) mới có thể sửa đổi các thanh ghi. Hãy tưởng tượng sự tàn phá một người dùng tiến trình có thể phá hỏng1 nếu nó có thể tự ý thay đổi thanh ghi cơ sở trong khi

[1] Có điều gì khác ngoài “sự tàn phá” có thể bị “phá hủy” không? [W17]

BÊN TRONG : CẤU TRÚC DỮ LIỆU - DANH SÁCH MIỄN PHÍ

Hệ điều hành phải theo dõi phần nào của bộ nhớ trống không được sử dụng, để có thể cấp phát bộ nhớ cho các tiến trình. Nhiều cấu trúc dữ liệu khác nhau tất nhiên có thể được sử dụng cho một nhiệm vụ như vậy; đơn giản nhất (mà chúng tôi sẽ giả định đây) là một danh sách miễn phí, đơn giản là danh sách các phạm vi của vật lý bộ nhớ hiện không được sử dụng.

yêu cầu phần cứng	Ghi chú
Chế độ đặc quyền	Cần thiết để ngăn các quy trình ở chế độ người dùng thực hiện các hoạt động đặc quyền
Đăng ký cơ sở / giới hạn	Cần cặp thanh ghi trên mỗi CPU để hỗ trợ dịch địa chỉ và kiểm tra giới hạn
Khả năng dịch các địa chỉ ảo Circuitry để thực hiện các bản dịch và kiểm tra các giới hạn; và kiểm tra xem có trong giới hạn không	trong trường hợp này, khá đơn giản
(Các) hướng dẫn dành riêng cho cập nhật cơ sở / giới hạn trước khi cho phép chương trình người dùng chạy	Hệ điều hành phải có thể đặt các giá trị này
(Các) hướng dẫn đặc quyền để đăng ký HĐH phải có khả năng cho phần cứng biết xử lý ngoại lệ	mã để chạy nếu ngoại lệ xảy ra
Khả năng nâng cao ngoại lệ	Khi các quy trình cố gắng truy cập các hướng dẫn đặc quyền hoặc bộ nhớ vượt quá giới hạn

Hình 15.3: Di dời động: Yêu cầu phần cứng

đang chạy. Hãy tưởng tượng nó! Và sau đó nhanh chóng loại bỏ những suy nghĩ đen tối như vậy khỏi tâm trí của bạn, vì chúng là thứ ghê rợn tạo nên những cơn ác mộng.

Cuối cùng, CPU phải có khả năng tạo ra các ngoại lệ trong các tình huống mà chương trình người dùng cố gắng truy cập bộ nhớ một cách bất hợp pháp (với địa chỉ "nằm ngoài giới hạn"); trong trường hợp này, CPU sẽ ngừng thực thi chương trình người dùng và sắp xếp để chạy trình xử lý ngoại lệ "nằm ngoài giới hạn" của hệ điều hành. Sau đó, trình xử lý hệ điều hành có thể tìm ra cách phản ứng, trong trường hợp này có thể kết thúc quá trình. Tương tự, nếu một chương trình người dùng cố gắng thay đổi các giá trị của thanh ghi cơ sở và giới hạn (đặc quyền), CPU sẽ tạo ra một ngoại lệ và chạy trình xử lý "đã cố gắng thực hiện một hoạt động đặc quyền khi ở chế độ người dùng". CPU cũng phải cung cấp một phương pháp để thông báo cho nó về vị trí của các bộ xử lý này; do đó cần thêm một vài hướng dẫn đặc quyền.

15.5 Các vấn đề về hệ điều hành

Cũng giống như phần cứng cung cấp các tính năng mới để hỗ trợ động relo cation, hệ điều hành hiện có các vấn đề mới mà nó phải xử lý; sự kết hợp giữa hỗ trợ phần cứng và quản lý hệ điều hành dẫn đến việc triển khai một bộ nhớ ảo đơn giản. Cụ thể, có một số đoạn quan trọng mà hệ điều hành phải tham gia để triển khai phiên bản cơ sở và giới hạn của bộ nhớ ảo.

Đầu tiên, HĐH phải thực hiện hành động khi một tiến trình được tạo, tìm kiếm không gian cho không gian địa chỉ của nó trong bộ nhớ. May mắn thay, với giả định của chúng tôi rằng mỗi không gian địa chỉ (a) nhỏ hơn kích thước của bộ nhớ vật lý và (b) cùng kích thước, điều này khá dễ dàng đối với hệ điều hành; nó có thể đơn giản xem bộ nhớ vật lý như một mảng các khe và theo dõi xem từng khe trống hay đang được sử dụng. Khi một tiến trình mới được tạo, HĐH sẽ phải tìm kiếm cấu trúc dữ liệu (thường được gọi là danh sách miễn phí) để tìm chỗ cho không gian địa chỉ mới và sau đó đánh dấu nó đã được sử dụng. Với không gian địa chỉ có kích thước thay đổi, cuộc sống phức tạp hơn, nhưng chúng tôi sẽ để lại mối quan tâm đó cho các chương sau.

Yêu cầu hệ điều hành	Ghi chú
Quản lý bộ nhớ	Cần cấp phát bộ nhớ cho các tiến trình mới; Lấy lại bộ nhớ từ các quy trình đã kết thúc; Nói chung quản lý bộ nhớ thông qua danh sách miễn phí
Quản lý cơ sở / giới hạn	Phải đặt cơ sở / giới hạn đúng cách khi chuyển đổi ngữ cảnh
Xử lý ngoại lệ	Mã để chạy khi phát sinh ngoại lệ; hành động có khả năng là chấm dứt quá trình vi phạm

Hình 15.4: Di dời động: Trách nhiệm của hệ điều hành

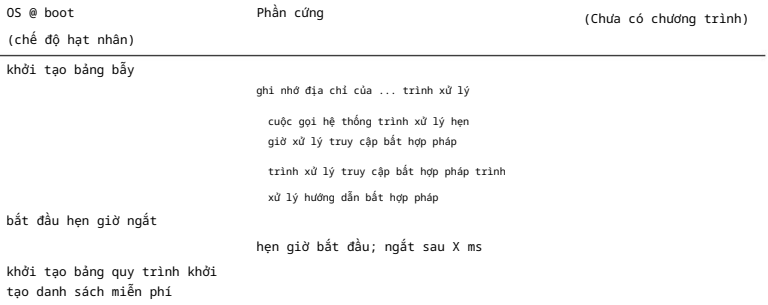
Hãy xem một ví dụ. Trong hình 15.2 (trang 5), bạn có thể thấy hệ điều hành sử dụng khe đầu tiên của bộ nhớ vật lý cho chính nó và nó đã được chuyển vị trí quy trình từ ví dụ trên vào vị trí bắt đầu tại địa chỉ ory mem vật lý 32 KB. Hai khe còn lại miễn phí (16 KB-32 KB và 48 KB 64 KB); do đó, danh sách miễn phí nên bao gồm hai mục này.

Thứ hai, hệ điều hành phải thực hiện một số công việc khi quá trình kết thúc (ví dụ: khi nó thoát ra một cách duyên dáng hoặc bị giết một cách cưỡng bức vì nó hoạt động sai), lấy lại tất cả bộ nhớ của nó để sử dụng trong các quy trình khác hoặc hệ điều hành. Trên chấm dứt quá trình, hệ điều hành do đó đặt bộ nhớ của nó trở lại miễn phí liệt kê và xóa mọi cấu trúc dữ liệu liên quan nếu cần.

Thứ ba, hệ điều hành cũng phải thực hiện một số bước bổ sung khi ngữ cảnh chuyển đổi xảy ra. Chỉ có một cặp thanh ghi cơ sở và giới hạn trên mỗi Xét cho cùng, CPU và giá trị của chúng khác nhau đối với từng chương trình đang chạy, vì mỗi chương trình được tải tại một địa chỉ vật lý khác trong bộ nhớ. Do đó, Hệ điều hành phải lưu và khôi phục cặp cơ sở và giới hạn khi nó chuyển sang hai quá trình. Cụ thể, khi hệ điều hành quyết định ngừng chạy một điểm chuyên nghiệp, nó phải lưu các giá trị của thanh ghi cơ sở và giới hạn vào bộ nhớ, trong một số cấu trúc cho mỗi quá trình, chẳng hạn như cấu trúc quá trình hoặc quá trình khối điều khiển (PCB). Tương tự, khi hệ điều hành tiếp tục một quá trình đang chạy (hoặc chạy nó lần đầu tiên), nó phải đặt các giá trị của cơ sở và giới hạn trên CPU đến các giá trị chính xác cho quá trình này.

Chúng ta nên lưu ý rằng khi một quá trình bị dừng (tức là không chạy), nó sẽ Hệ điều hành có thể di chuyển một không gian địa chỉ từ vị trí này trong mem ory sang vị trí khác khá dễ dàng. Để di chuyển không gian địa chỉ của quy trình, hệ điều hành đầu tiên mô tả quy trình; sau đó, hệ điều hành sao chép không gian địa chỉ từ vị trí hiện tại đến vị trí mới; cuối cùng, hệ điều hành cập nhật thanh ghi cơ sở (trong cấu trúc tiến trình) để trỏ đến vị trí mới. Khi nào quá trình được tiếp tục, thanh ghi cơ sở (mới) của nó được khôi phục và nó bắt đầu đang hoạt động trở lại, không biết rằng các hướng dẫn và dữ liệu của nó giờ đã ở một vị trí hoàn toàn mới trong bộ nhớ.

Thứ tư, Hệ điều hành phải cung cấp các trình xử lý ngoại lệ hoặc các chức năng để được gọi là, như đã thảo luận ở trên; hệ điều hành cài đặt các trình xử lý này tại thời điểm khởi động (thông qua hướng dẫn đặc quyền). Ví dụ, nếu một tiến trình cố gắng truy cập mem ory bên ngoài giới hạn của nó, CPU sẽ đưa ra một ngoại lệ; hệ điều hành phải là chuẩn bị hành động khi một ngoại lệ như vậy phát sinh. Phản ứng chung của hệ điều hành sẽ là một trong những sự thù địch: nó có thể sẽ chấm dứt hành vi vi phạm quá trình. Hệ điều hành phải được bảo vệ cao đối với máy mà nó đang chạy, và do đó nó không cần đến một quá trình cố gắng truy cập vào bộ nhớ hoặc



Hình 15.5: Thực thi trực tiếp có giới hạn (Chuyển vị trí động) @ Boot

thực hiện các hướng dẫn mà nó không nên. Tạm biệt, quy trình hoạt động sai; thật vui khi biết bạn.

Hình 15.5 và 15.6 (trang 12) minh họa phần lớn sự tương tác giữa phần cứng / hệ điều hành trong một dòng thời gian. Hình đầu tiên cho thấy hệ điều hành làm gì tại thời điểm khởi động để máy sẵn sàng sử dụng và hình thứ hai cho thấy điều gì sẽ xảy ra khi một tiến trình (Process A) bắt đầu chạy; lưu ý cách phần cứng xử lý các transla bộ nhớ mà không có sự can thiệp của hệ điều hành. Tại một số thời điểm (giữa hình thứ hai), một ngắt bộ đếm thời gian xảy ra và hệ điều hành chuyển sang Quy trình B, quá trình này thực hiện "tải xấu" (đến một địa chỉ bộ nhớ bất hợp pháp); tại thời điểm đó, OS phải tham gia, kết thúc tiến trình và dọn dẹp bằng cách giải phóng bộ nhớ của B và xóa mục nhập của nó khỏi bảng tiến trình. Như bạn có thể thấy từ các hình, chúng tôi vẫn đang tuân theo phương pháp cơ bản là thực hiện trực tiếp hạn chế. Trong hầu hết các trường hợp, HĐH chỉ thiết lập phần cứng một cách thích hợp và cho phép quá trình chạy trực tiếp trên CPU; chỉ khi quá trình hoạt động sai thì HĐH mới phải tham gia.

15.6 Tóm tắt

Trong chương này, chúng tôi đã mở rộng khái niệm về cảnh báo exe trực tiếp có giới hạn với một cơ chế cụ thể được sử dụng trong bộ nhớ ảo, được gọi là bản dịch trang phục quảng cáo. Với tính năng dịch địa chỉ, hệ điều hành có thể kiểm soát mọi truy cập bộ nhớ từ một tiến trình, đảm bảo các truy cập nằm trong giới hạn của không gian địa chỉ. Chìa khóa cho hiệu quả của kỹ thuật này là hỗ trợ phần cứng, thực hiện việc dịch nhanh chóng cho mỗi điểm ac, biến các địa chỉ ảo (chế độ xem của bộ nhớ) thành các địa chỉ vật lý (chế độ xem thực tế). Tất cả những điều này được thực hiện theo cách mà là phụ huynh chuyển giao cho quá trình đã được di dời; quá trình này không biết rằng các tham chiếu bộ nhớ của nó đang được dịch, tạo ra một ảo ảnh tuyệt vời.

Chúng tôi cũng đã thấy một dạng ảo hóa cụ thể, được gọi là cơ sở và giới hạn hoặc di dời động. Ảo hóa cơ sở và giới hạn khá hiệu quả, vì chỉ cần thêm một chút logic phần cứng để thêm

OS @ run (chế độ hạt nhân)	Phần cứng	Chương trình (chế độ người dùng)
Để bắt đầu quy trình A:		
cấp phát mục nhập trong bảng quy trình		
cấp phát bộ nhớ cho các thanh		
ghi cơ sở / giới hạn bộ quy trình trả về từ bẫy (thành A)		
	khôi phục các thanh ghi của A chuyển sang chế độ người dùng	
	dùng chuyển sang PC (ban đầu) của A	
		Quá trình A chạy
		Hướng dẫn tìm nạp
	dịch địa chỉ ảo thực hiện tìm nạp	
		Thực hiện hướng dẫn
	nếu tải / lưu trữ rõ ràng: đảm bảo địa chỉ là hợp pháp dịch địa chỉ ảo thực hiện tải / lưu trữ	
		(A chạy ...)
	Bộ hẹn giờ ngắt đi chuyển sang chế độ hạt nhân	
	nhảy tới trình xử lý	
Xử lý bộ hẹn giờ		
quyết định: dừng A, chạy B chuyển đổi cuộc gọi () quy trình lưu regs (A) vào proc-struct (A) (bao gồm cơ sở / giới hạn) khôi phục regs (B) từ proc-struct (B) (bao gồm cơ sở / giới hạn) return-from-trap (vào B)		
	khôi phục các thanh ghi của B chuyển sang chế độ người dùng	
	chuyển sang PC của B	
		Quá trình B chạy
		Thực thi tải không tốt
	Tải quá giới hạn; chuyển sang chế độ hạt nhân nhảy đến	
	trình xử lý bẫy	
Xử lý cái bẫy quyết định giết quy trình B giải quyết cấp bộ nhớ trống của B mục nhập của B trong bảng quy trình		

Hình 15.6: Thực hiện trực tiếp có giới hạn (Chuyển vị trí động) @ Runtime

đăng ký cơ sở đến địa chỉ ảo và kiểm tra xem địa chỉ được tạo bởi quá trình có nằm trong giới hạn hay không. Cơ sở và giới hạn cũng cung cấp sự bảo vệ; Hệ điều hành và phần cứng kết hợp để đảm bảo không có quá trình nào có thể tạo ra các tham chiếu bộ nhớ bên ngoài không gian địa chỉ của chính nó. Bảo vệ chắc chắn là một trong những mục tiêu quan trọng nhất của HĐH; nếu không có nó, hệ điều hành không thể kiểm soát máy (nếu các tiến trình được tự do ghi đè bộ nhớ, chúng có thể dễ dàng làm những việc khó chịu như ghi đè bảng bẫy và chiếm quyền điều khiển hệ thống).

Thật không may, kỹ thuật đơn giản của việc tái định cư động này lại không hiệu quả. Ví dụ, như bạn có thể thấy trong Hình 15.2 (trang 5), quá trình được di dời đang sử dụng bộ nhớ vật lý từ 32 KB đến 48 KB; bao giờ hết, bởi vì ngăn xếp quy trình và đống không quá lớn, tất cả không gian giữa hai quy trình này đơn giản là lãng phí. Loại lãng phí này thường được gọi là phân mảnh cuối cấp, vì không gian bên trong đơn vị được cấp phát không được sử dụng hết (tức là bị phân mảnh) và do đó bị lãng phí. Trong cách tiếp cận hiện tại của chúng tôi, mặc dù có thể có đủ bộ nhớ vật lý cho nhiều quy trình hơn, nhưng chúng tôi bị hạn chế một cách đáng kể trong việc đặt một không gian địa chỉ trong một khe có kích thước cố định và do đó có thể phát sinh phân mảnh nội bộ[2] . Vì vậy, chúng ta sẽ cần nhiều máy móc thiết bị tinh vi hơn, để cố gắng sử dụng bộ nhớ vật lý tốt hơn và tránh phân mảnh nội bộ. Nỗ lực đầu tiên của chúng tôi sẽ là một sự khái quát nhẹ về cơ sở và giới hạn được gọi là phân đoạn, mà chúng ta sẽ thảo luận tiếp theo.

[2] Thay vào đó, giải pháp khác 2A có thể đặt một ngăn xếp có kích thước cố định trong không gian địa chỉ, ngay bên dưới vùng mã và một đống đang phát triển bên dưới vùng đó. Tuy nhiên, điều này hạn chế tính linh hoạt bằng cách thực hiện các cuộc gọi hàm đệ quy và lồng nhau sâu sắc là thách thức và do đó chúng tôi hy vọng sẽ tránh được.

Người giới thiệu

[M65] “On Dynamic Program Relocation” của WC McGee. Tạp chí Hệ thống IBM, Tập 4: 3, 1965, trang 184–199. Bài báo này là một bản tóm tắt tốt đẹp về công việc ban đầu về tái định cư động, cũng như một số điều cơ bản về tái định cư tĩnh.

[P90] “Định vị lại bộ tải cho các tệp thực thi MS-DOS .EXE” của Kenneth DA Pillay. Bộ xử lý vi mô & kho lưu trữ Hệ thống vi mô, Tập 14: 7 (tháng 9 năm 1990). Ví dụ về bộ tải định vị lại cho MS-DOS. Không phải là cái đầu tiên, mà chỉ là một ví dụ tương đối hiện đại về cách thức hoạt động của một hệ thống như vậy.

[SS74] “Bảo vệ thông tin trong hệ thống máy tính” của J. Saltzer và M. Schroeder.
CACM, tháng 7 năm 1974. Từ bài báo này: “Các khái niệm về thanh ghi cơ sở và giới hạn và các bộ mô tả thông dịch phần cứng đã xuất hiện, rõ ràng là độc lập, trong khoảng thời gian từ 1957 đến 1959 trên ba dự án với các mục tiêu đa dạng. Tại MIT, McCarthy đề xuất ý tưởng cơ sở và giới hạn như một phần của hệ thống bảo vệ bộ nhớ cần thiết để làm cho việc chia sẻ thời gian trở nên khả thi. IBM đã độc lập phát triển thanh ghi cơ sở và giới hạn như một cơ chế cho phép lập trình đa chương trình đáng tin cậy của hệ thống máy tính Stretch (7030). Tại Burroughs, R. Barton gợi ý rằng các bộ mô tả thông dịch bằng phần cứng sẽ cung cấp hỗ trợ trực tiếp cho các quy tắc phạm vi đặt tên của các ngôn ngữ cấp cao hơn trong hệ thống máy tính B5000 ”. Chúng tôi tìm thấy câu trích dẫn này trên các trang lịch sử thú vị của Mark S Aceman [S04]; xem chúng để biết thêm thông tin.

[S04] “Hỗ trợ cuộc gọi hệ thống” của Mark Sesyman. Tháng 5 năm 2004. people.cs.clemson.edu/ ¨mark / syscall.html.
Lịch sử hỗ trợ cuộc gọi hệ thống gọn gàng. S Motherman cũng đã thu thập một số lịch sử ban đầu về các mục như ngắt và các khía cạnh thú vị khác của lịch sử máy tính. Xem các trang web của anh ấy để biết thêm chi tiết.

[WL + 93] “Cách ly lỗi dựa trên phần mềm hiệu quả” của Robert Wahbe, Steven Lucco, Thomas E. Anderson, Susan L. Graham. SOSP '93. Một bài báo tuyệt vời về cách bạn có thể sử dụng hỗ trợ trình biên dịch để ràng buộc các tham chiếu bộ nhớ từ một chương trình mà không cần hỗ trợ phần cứng. Bài báo đã khơi dậy mối quan tâm mới đến các kỹ thuật phần mềm để cô lập các tham chiếu bộ nhớ.

[W17] Câu trả lời cho chú thích cuối trang: "Có điều gì khác ngoài sự tàn phá có thể bị phá hủy không?" của Waciuma Wanjohi. Tháng 10 năm 2017. Thật ngạc nhiên, độc giả dám nghĩ dám làm này đã tìm thấy câu trả lời thông qua công cụ xem Ngram của google (có tại URL sau: http://books.google.com.vn/ngrams).
Câu trả lời là nhờ ông Wanjohi: “Chỉ từ khoảng năm 1970, 'sự tàn phá' mới trở nên phổ biến hơn 'sự báo thù'. Vào những năm 1800, từ tàn phá hầu như luôn được theo sau bởi 'sự báo thù của anh ấy / họ'. " Rõ ràng, khi bạn phá hủy, bạn sẽ không tốt, nhưng ít nhất những người phá hủy có một số lựa chọn bây giờ.

Bài tập về nhà (Mô phỏng)

Chương trình tái định cư.py cho phép bạn xem cách chuyển đổi địa chỉ được thực hiện trong một hệ thống có thanh ghi cơ sở và giới hạn. Xem README để biết chi tiết.

Câu hỏi

1. Chạy với các hạt giống 1, 2 và 3 và tính toán xem mỗi trang phục quảng cáo ảo do quá trình tạo ra có nằm trong giới hạn hay không. Nếu trong giới hạn, hãy tính bản dịch.
2. Chạy với các cờ sau: -s 0 -n 10. Bạn đặt -l (thanh ghi giới hạn) thành giá trị nào để đảm bảo rằng tất cả các địa chỉ ảo được tạo đều nằm trong giới hạn?
3. Chạy với các cờ sau: -s 1 -n 10 -l 100. Giá trị tối đa mà cơ sở có thể được đặt thành là bao nhiêu, sao cho không gian địa chỉ vẫn nằm gọn trong bộ nhớ vật lý?
4. Chạy một số vấn đề tương tự ở trên, nhưng với không gian địa chỉ lớn hơn (-a) và bộ nhớ vật lý (-p).
5. Phần nào của địa chỉ ảo được tạo ngẫu nhiên là hợp lệ, như một hàm của giá trị của thanh ghi giới hạn? Tạo một biểu đồ từ việc chạy với các hạt ngẫu nhiên khác nhau, với các giá trị giới hạn được đặt từ 0 đến kích thước tối đa của không gian địa chỉ.