18

Phân trang: Giới thiệu

Đôi khi người ta nói rằng hệ điều hành sử dụng một trong hai cách tiếp cận khi giải quyết hầu hết mọi vấn đề về quản lý không gian. Cách tiếp cận đầu tiên là cắt nhỏ mọi thứ thành các phần có kích thước thay đổi, như chúng ta đã thấy với phân đoạn trong bộ nhớ ảo. Thật không may, giải pháp này có những ràng buộc khó khăn cố hữu. Đặc biệt, khi chia không gian thành các phần có kích thước khác nhau, bản thân không gian đó có thể bị phân mảnh và do đó việc phân bổ trở nên khó khăn hơn theo thời gian.

Vì vậy, có thể đáng xem xét cách tiếp cận thứ hai: cắt không gian thành các mảnh có kích thước cố định. Trong bộ nhớ ảo, chúng tôi gọi đây là phân trang ý tưởng và nó quay trở lại một hệ thống sơ khai và quan trọng, Atlas [KE + 62, L78]. Thay vì chia không gian địa chỉ của quy trình thành một số phân đoạn logic có kích thước thay đổi (ví dụ: mã, đống, ngăn xếp), chúng tôi chia nó thành các đơn vị có kích thước cố định, mỗi đơn vị chúng tôi gọi là một trang. Tương ứng, chúng ta xem bộ nhớ vật lý như một mảng các khe có kích thước cố định được gọi là khung trang; mỗi khung này có thể chứa một trang bộ nhớ ảo. Thách thức của chúng tôi:

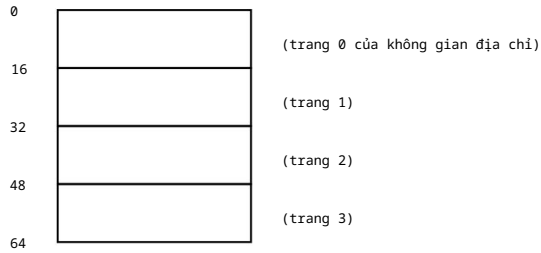
CRUX :
CÁCH ĐÁNH GIÁ BỘ NHỚ VỚI CÁC TRANG

Làm thế nào chúng ta có thể ảo hóa bộ nhớ với các trang để tránh lỗi phân đoạn? Các kỹ thuật cơ bản là gì? Làm thế nào để chúng tôi làm cho những kỹ thuật đó hoạt động tốt, với chi phí tối thiểu về không gian và thời gian?

18.1 Một ví dụ đơn giản và tổng quan

Để giúp cách tiếp cận này rõ ràng hơn, hãy minh họa nó bằng một ví dụ đơn giản. Hình 18.1 (trang 2) trình bày một ví dụ về một không gian địa chỉ nhỏ, chỉ có tổng kích thước 64 byte, với bốn trang 16 byte (trang ảo 0, 1, 2 và 3). Tất nhiên, không gian địa chỉ thực lớn hơn nhiều, thường là 32 bit và do đó là 4 GB không gian địa chỉ, hoặc thậm chí 64 bit1 ; trong cuốn sách, chúng tôi thường sử dụng các ví dụ nhỏ để làm cho chúng dễ hiểu hơn.

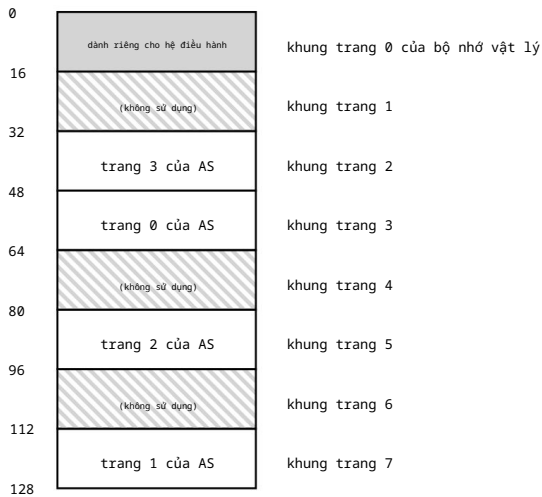
1 Không gian địa chỉ 64-bit thật khó tưởng tượng, nó lớn đến mức đáng kinh ngạc. Một phép tương tự có thể hữu ích: nếu bạn nghĩ không gian địa chỉ 32 bit có kích thước bằng một sân tennis, thì không gian địa chỉ 64 bit có kích thước tương đương với Châu Âu (!).



Hình 18.1: Không gian địa chỉ 64 byte đơn giản

Bộ nhớ vật lý, như thể hiện trong Hình 18.2, cũng bao gồm một số khe cắm có kích thư ớc cố định, trong trư ờng hợp này là tám khung trang (tạo cho bộ nhớ vật lý 128 byte, cũng nhỏ đến mức kỳ cục). Như bạn có thể thấy trong sơ đồ, các trang của không gian địa chỉ ảo đã đư ợc đặt ở các điểm khác nhau trong bộ nhớ vật lý; sơ đồ cũng cho thấy hệ điều hành sử dụng một số bộ nhớ vật lý cho chính nó.

Phân trang, như chúng ta sẽ thấy, có một số lợi thế so với các cách tiếp cận trư ớc đây của chúng tôi. Có lẽ cải tiến quan trọng nhất sẽ là tính linh hoạt: với cách tiếp cận phân trang đư ợc phát triển đầy đủ, hệ thống sẽ có thể hỗ trợ trừu tư ợng hóa không gian địa chỉ một cách hiệu quả, bất kể quy trình sử dụng không gian địa chỉ như thế nào; chẳng hạn, chúng tôi sẽ không đư a ra giả định về hư ớng phát triển của đống và ngăn xếp cũng như cách chúng đư ợc sử dụng.



Hình 18.2: Không gian địa chỉ 64 byte trong bộ nhớ vật lý 128 byte

Một ưu điểm khác là sự đơn giản của việc quản lý không gian trống mà trang có sẵn. Ví dụ, khi hệ điều hành muốn đặt không gian địa chỉ 64 byte nhỏ bé của chúng tôi vào bộ nhớ vật lý tám trang của chúng tôi, nó chỉ đơn giản là tìm thấy bốn trang miễn phí; có lẽ hệ điều hành giữ một danh sách miễn phí tất cả các trang miễn phí cho việc này, và chỉ lấy bốn trang miễn phí đầu tiên trong danh sách này. Trong ví dụ, HĐH đã đặt trang ảo 0 của không gian địa chỉ (AS) trong khung vật lý 3, trang ảo 1 của AS trong khung vật lý 7, trang 2 trong khung 5 và trang 3 trong khung 2. Các khung trang 1 , 4 và 6 hiện đang miễn phí.

Để ghi lại vị trí đặt mỗi trang ảo của không gian địa chỉ trong bộ nhớ vật lý, hệ điều hành thư ờng giữ một cấu trúc dữ liệu cho mỗi quá trình đư ợc gọi là bảng trang. Vai trò chính của bảng trang là lư u trữ các bản dịch địa chỉ cho mỗi trang ảo của không gian địa chỉ, do đó cho chúng ta biết vị trí của mỗi trang trong bộ nhớ vật lý.
Đối với ví dụ đơn giản của chúng tôi (Hình 18.2, trang 2), bảng trang sẽ có bốn mục sau: (Trang ảo 0 Khung vật lý 3), (VP 1 PF 7), (VP 2 PF 5), và (VP 3 PF 2).

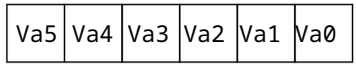
Điều quan trọng cần nhớ là bảng trang này là một cấu trúc dữ liệu cho mỗi quá trình (hầu hết các cấu trúc bảng trang mà chúng ta thảo luận là các cấu trúc struc cho mỗi quá trình; một ngoại lệ mà chúng tôi sẽ đề cập là bảng trang đảo ngư ợc). Nếu một quy trình khác đư ợc chạy trong ví dụ của chúng tôi ở trên, OS sẽ phải quản lý một bảng trang khác cho nó, vì các trang ảo của nó rõ ràng là ánh xạ đến các trang vật lý khác nhau (mô-đun bất kỳ chia sẻ nào đang diễn ra).

Bây giờ, chúng ta đã biết đủ để thực hiện một ví dụ dịch địa chỉ. Hãy tư ởng tư ợng quá trình với không gian địa chỉ nhỏ bé (64 byte) đang hình thành một truy cập bộ nhớ:

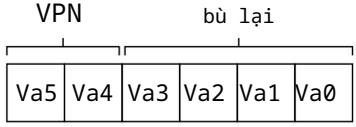
```
movl <địa chỉ ảo>,% eax
```

Cụ thể, chúng ta hãy chú ý đến việc tải dữ liệu rõ ràng từ địa chỉ <địa chỉ ảo> vào thanh ghi eax (và do đó bỏ qua việc tìm nạp lệnh đã xảy ra trư ớc đó).

Để dịch địa chỉ ảo này mà quá trình tạo ra, trư ớc tiên chúng ta phải chia nó thành hai thành phần: số trang ảo (VPN) và phần bù bên trong trang. Đối với ví dụ này, vì không gian địa chỉ ảo của tiến trình là 64 byte, chúng ta cần tổng cộng 6 bit cho địa chỉ ảo của mình ($2^6 = 64$). Do đó, địa chỉ ảo của chúng ta có thể đư ợc khái niệm như sau:



Trong sơ đồ này, Va5 là bit bậc cao nhất của địa chỉ ảo và Va0 là bit bậc thấp nhất. Bởi vì chúng tôi biết kích thư ớc trang (16 byte), chúng tôi có thể chia thêm địa chỉ ảo như sau:

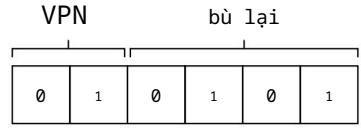


Kích thước trang là 16 byte trong không gian địa chỉ 64 byte; do đó chúng ta cần phải có thể chọn 4 trang và 2 bit trên cùng của địa chỉ thực hiện điều đó. Như vậy, chúng ta có số trang ảo (VPN) 2 bit. Các bit còn lại cho biết cho chúng tôi byte nào của trang mà chúng tôi quan tâm, 4 bit trong trường hợp này; chúng tôi gọi đây là sự bù đắp.

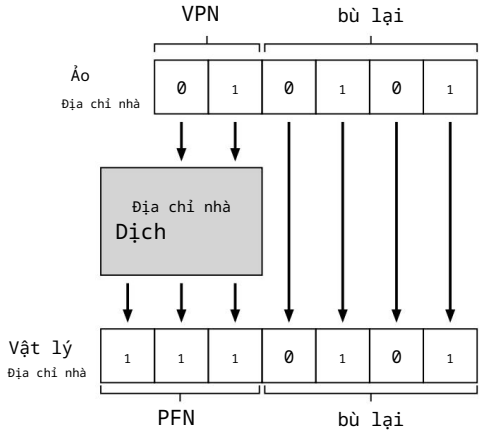
Khi một quy trình tạo ra một địa chỉ ảo, hệ điều hành và phần cứng phải kết hợp để dịch nó thành một địa chỉ vật lý có ý nghĩa. Đối với rất nhiều, chúng ta hãy giả sử tải ở trên là đến địa chỉ ảo 21:

```
movl 21,%eax
```

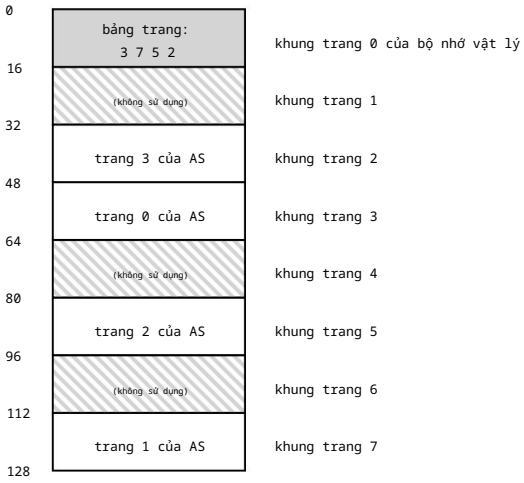
Chuyển "21" thành dạng nhị phân, chúng tôi nhận được "010101" và do đó chúng tôi có thể lấy địa chỉ ảo này và xem cách nó chia thành một trang ảo số (VPN) và bù đắp:



Do đó, địa chỉ ảo "21" nằm trên byte thứ 5 ("0101") của địa chỉ ảo trang "01" (hoặc 1). Với số trang ảo của chúng tôi, bây giờ chúng tôi có thể lập chỉ mục bảng trang và tìm khung vật lý trang ảo 1 nằm bên trong. Trong bảng trang phía trên số khung vật lý (PFN) (đôi khi được gọi là số trang vật lý hoặc PPN) là 7 (nhị phân 111). Do đó, chúng ta có thể dịch địa chỉ ảo này bằng cách thay thế VPN bằng PFN và sau đó cấp tải cho bộ nhớ vật lý (Hình 18.3).



Hình 18.3: Quy trình dịch địa chỉ



Hình 18.4: Ví dụ: Bảng trang trong bộ nhớ vật lý hạt nhân

Lưu ý rằng độ lệch vẫn giữ nguyên (nghĩa là nó không được dịch), bởi vì độ lệch chỉ cho chúng ta biết byte nào trong trang mà chúng ta muốn. Địa chỉ vật lý cuối cùng của chúng tôi là 1110101 (117 trong hệ thập phân) và chính xác là nơi chúng tôi muốn tải của mình để tìm nạp dữ liệu từ đó (Hình 18.2, trang 2).

Với tổng quan cơ bản này trong tâm trí, bây giờ chúng ta có thể hỏi (và hy vọng có thể trả lời) một số câu hỏi cơ bản mà bạn có thể có về phân trang. Ví dụ, những bảng trang này được lưu trữ ở đâu? Nội dung tiêu biểu của bảng trang là gì và các bảng lớn như thế nào? Phân trang có làm cho hệ thống (quá) chậm không? Những câu hỏi này và các câu hỏi hấp dẫn khác được trả lời, ít nhất một phần, trong văn bản bên dưới. Đọc tiếp!

18.2 Các bảng trang được lưu trữ ở đâu?

Bảng trang có thể lớn khủng khiếp, lớn hơn nhiều so với bảng phân đoạn nhỏ hoặc cặp cơ sở / giới hạn mà chúng ta đã thảo luận trước đây. Ví dụ: hãy tưởng tượng một không gian địa chỉ 32-bit điển hình, với các trang 4KB. Chiếc váy quảng cáo ảo này chia thành VPN 20 bit và bù 12 bit (nhớ lại rằng 10 bit sẽ cần thiết cho kích thước trang 1KB và chỉ cần thêm hai bit nữa để có được 4KB).

VPN 20 bit ngụ ý rằng có 2^{20} các bản dịch mà Hệ điều hành sẽ phải quản lý cho mỗi quá trình (khoảng một triệu); giả sử chúng ta cần 4 byte cho mỗi mục nhập bảng trang (PTE) để giữ bản dịch vật lý cùng với bất kỳ nội dung hữu ích nào khác, chúng ta sẽ nhận được 4MB bộ nhớ khổng lồ cần thiết cho mỗi bảng trang! Đó là khá lớn. Bây giờ hãy tưởng tượng có 100 tiến trình đang chạy: điều này có nghĩa là hệ điều hành sẽ cần 400MB bộ nhớ chỉ cho tất cả các bản dịch địa chỉ đó! Ngay cả trong thời kỳ hiện đại, nơi

ASIDE: CẤU TRÚC DỮ LIỆU - BẢNG TRANG Một trong những cấu trúc dữ liệu quan trọng nhất trong hệ thống con quản lý bộ nhớ của HĐH hiện đại là bảng trang. Nói chung, một bảng trang lưu trữ các bản dịch địa chỉ ảo sang vật lý, do đó cho hệ thống biết vị trí của mỗi trang trong không gian địa chỉ thực sự nằm trong bộ nhớ vật lý. Bởi vì mỗi không gian địa chỉ yêu cầu các bản dịch như vậy, trong gen eral có một bảng trang cho mỗi quá trình trong hệ thống. Cấu trúc chính xác của bảng trang được xác định bởi phần cứng (hệ thống cũ hơn) hoặc có thể được quản lý linh hoạt hơn bởi hệ điều hành (hệ thống hiện đại).

máy có bộ nhớ hàng gigabyte, có vẻ hơi điên rồ khi sử dụng một phần lớn bộ nhớ chỉ để dịch, phải không? Và chúng tôi thậm chí sẽ không nghĩ về việc một bảng trang như vậy sẽ lớn như thế nào đối với không gian địa chỉ 64-bit; điều đó sẽ quá khủng khiếp và có thể khiến bạn sợ hãi hoàn toàn.

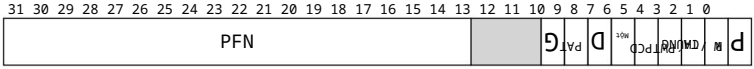
Bởi vì bảng trang quá lớn, chúng tôi không giữ bất kỳ thiết bị cứng trên chip đặc biệt nào trong MMU để lưu trữ bảng trang của quy trình hiện đang chạy.
Thay vào đó, chúng tôi lưu trữ bảng trang cho mỗi quá trình trong bộ nhớ ở đâu đó. Bây giờ, hãy giả sử rằng các bảng trang nằm trong bộ nhớ vật lý mà hệ điều hành quản lý; sau này chúng ta sẽ thấy rằng bản thân phần lớn bộ nhớ HĐH có thể được điều chỉnh và do đó các bảng trang có thể được lưu trữ trong bộ nhớ ảo HĐH (và thậm chí được hoán đổi sang đĩa), nhưng điều đó quá khó hiểu ngay bây giờ, vì vậy chúng ta sẽ tìm hiểu nó . Trong Hình 18.4 (trang 5) là hình ảnh của một bảng trang trong bộ nhớ hệ điều hành; xem tập hợp các bản dịch nhỏ trong đó?

18.3 Thực sự có gì trong bảng trang?

Hãy nói một chút về tổ chức bảng trang. Bảng trang chỉ là một cấu trúc dữ liệu được sử dụng để ánh xạ địa chỉ ảo (hoặc thực sự, số trang ảo) với địa chỉ thực (số khung vật lý). Do đó, bất kỳ cấu trúc dữ liệu nào cũng có thể hoạt động. Dạng đơn giản nhất được gọi là bảng trang tuyến tính, chỉ là một mảng. Hệ điều hành lập chỉ mục mảng theo số trang ảo (VPN) và tra cứu mục nhập bảng trang (PTE) tại chỉ mục đó để tìm số khung vật lý (PFN) mong muốn. Bây giờ, chúng ta sẽ giả sử cấu trúc tuyến tính đơn giản này; trong các chương sau, chúng tôi sẽ sử dụng các cấu trúc dữ liệu nâng cao hơn để giúp giải quyết một số vấn đề với phân trang.

Đối với nội dung của mỗi PTE, chúng ta có một số bit khác nhau trong đó đáng để hiểu ở một mức độ nào đó. Một bit hợp lệ là phổ biến để cho biết liệu bản dịch cụ thể có hợp lệ hay không; ví dụ: khi một chương trình bắt đầu chạy, nó sẽ có mã và đống ở một đầu của không gian địa chỉ và ngăn xếp ở đầu kia. Tất cả không gian chưa sử dụng ở giữa sẽ bị đánh dấu là không hợp lệ và nếu quá trình cố gắng truy cập bộ nhớ đó, nó sẽ tạo ra một cái bẫy đối với Hệ điều hành và có khả năng sẽ chấm dứt quá trình.

Vì vậy, bit hợp lệ là rất quan trọng để hỗ trợ một không gian địa chỉ thưa thớt; bằng cách chỉ cần đánh dấu tất cả các trang không sử dụng trong không gian địa chỉ là không hợp lệ, chúng tôi loại bỏ nhu cầu phân bổ khung vật lý cho các trang đó và do đó tiết kiệm rất nhiều bộ nhớ.



Hình 18.5: Một mục nhập bảng trang x86 (PTE)

Chúng tôi cũng có thể có các bit bảo vệ, cho biết liệu trang có thể đư ợc đọc từ, ghi vào hoặc thực thi từ đó hay không. Một lần nữa, việc truy cập một trang theo cách không đư ợc các bit này cho phép sẽ tạo ra một cái bẫy đối với Hệ điều hành.

Có một vài điều quan trọng khác như ng chúng ta sẽ không nói nhiều về vấn đề này bây giờ. Một bit hiện tại cho biết liệu trang này nằm trong bộ nhớ vật lý hay trên đĩa (tức là nó đã đư ợc hoán đổi). Chúng tôi sẽ nghiên cứu kỹ hơn về bộ máy này khi nghiên cứu cách hoán đổi các phần của không gian địa chỉ sang đĩa để hỗ trợ các không gian địa chỉ lớn hơn bộ nhớ vật lý; hoán đổi cho phép HĐH giải phóng bộ nhớ vật lý bằng cách di chuyển các trang hiếm khi đư ợc sử dụng vào đĩa. Một bit bẩn cũng phổ biến, cho biết liệu trang đã đư ợc sửa đổi kể từ khi nó đư ợc đư a vào bộ nhớ hay chư a.

Một bit tham chiếu (hay còn gọi là bit đư ợc truy cập) đôi khi đư ợc sử dụng để theo dõi xem một trang đã đư ợc truy cập hay chư a và hữu ích trong việc xác định trang nào phổ biến và do đó nên đư ợc lư u trong bộ nhớ; những kiến thức đó rất quan trọng trong quá trình thay thế trang, một chủ đề chúng tôi sẽ nghiên cứu rất chi tiết trong các chư ơng sau.

Hình 18.5 cho thấy một mục nhập bảng trang ví dụ từ x86 architec ture [I09]. Nó chứa một bit hiện tại (P); một bit đọc / ghi (R / W) xác định xem có đư ợc phép ghi vào trang này hay không; một bit ngư ời dùng / ngư ời giám sát (U / S) xác định xem các quy trình ở chế độ ngư ời dùng có thể truy cập trang hay không; một vài bit (PWT, PCD, PAT và G) xác định cách bộ nhớ đệm phần cứng hoạt động cho các trang này; một bit đư ợc truy cập (A) và một bit bẩn (D); và cuối cùng là số khung trang (PFN) của chính nó.

Đọc Hư ớng dẫn sử dụng kiến trúc Intel [I09] để biết thêm chi tiết về hỗ trợ trang x86. Đư ợc cảnh báo trư ớc, tuy nhiên; Việc đọc các hư ớng dẫn sử dụng như thế này, mặc dù khá nhiều thông tin (và chắc chắn cần thiết cho những ngư ời viết mã để sử dụng các bảng trang như vậy trong HĐH), lúc đầu có thể là một thách thức. Cần phải có một chút thận trọng và rất nhiều khát khao.

ASIDE: TẠI SAO KHÔNG CÓ BIT HỢP LỆ ?

Bạn có thể nhận thấy rằng trong ví dụ Intel, không có bit hiện tại và hợp lệ riêng biệt mà chỉ là bit hiện tại (P). Nếu bit đó đư ợc đặt (P = 1), điều đó có nghĩa là trang vừa hiện tại vừa hợp lệ. Nếu không (P = 0), điều đó có nghĩa là trang có thể không có trong bộ nhớ (như ng hợp lệ), hoặc có thể không hợp lệ. Quyền truy cập vào một trang có P = 0 sẽ kích hoạt một cái bẫy đối với Hệ điều hành; Hệ điều hành sau đó phải sử dụng các cấu trúc bổ sung mà nó lư u giữ để xác định xem trang có hợp lệ hay không (và do đó có lẽ nên đư ợc hoán đổi lại) hay không (và do đó chư ơng trình đang cố gắng truy cập bộ nhớ một cách bất hợp pháp). Sự thận trọng này là phổ biến trong phần cứng, thư ờng chỉ cung cấp một số tính năng tối thiểu mà hệ điều hành có thể xây dựng một dịch vụ đầy đủ.

18.4 Phân trang: Cũng quá chậm

Với các bảng trang trong bộ nhớ, chúng ta đã biết rằng chúng có thể quá lớn. Hóa ra, chúng cũng có thể làm mọi thứ chậm lại. Ví dụ, hãy xem hư ớng dẫn đơn giản của chúng tôi:

```
tháng 21,% eax
```

Một lần nữa, chúng ta hãy chỉ kiểm tra tham chiếu rõ ràng đến địa chỉ 21 và đừng lo lắng về việc tìm nạp lệnh. Trong ví dụ này, chúng tôi sẽ giả sử phần cứng thực hiện bản dịch cho chúng tôi. Để tìm nạp dữ liệu mong muốn, trư ớc tiên hệ thống phải dịch địa chỉ ảo (21) thành trang phục quảng cáo vật lý chính xác (117). Do đó, trư ớc khi tìm nạp dữ liệu từ địa chỉ 117, trư ớc tiên hệ thống phải tìm nạp mục nhập bảng trang thích hợp từ bảng trang của quy trình, thực hiện dịch và sau đó tải dữ liệu từ bộ nhớ vật lý.

Để làm như vậy, phần cứng phải biết vị trí của bảng trang cho quá trình hiện đang chạy. Bây giờ, hãy giả sử rằng một thanh ghi cơ sở bảng trang chứa địa chỉ vật lý của vị trí bắt đầu của bảng trang. Để tìm vị trí của PTE mong muốn, phần cứng sẽ thực hiện các chức năng sau:

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
PTEAddr = PageTableBaseRegister + (VPN * sizeof (PTE))
```

Trong ví dụ của chúng tôi, VPN MASK sẽ đư ợc đặt thành 0x30 (hex 30 hoặc nhị phân 110000) chọn ra các bit VPN từ địa chỉ ảo đầy đủ; SHIFT đư ợc đặt thành 4 (số bit trong phần bù), để chúng tôi di chuyển các bit VPN xuống để tạo thành số trang ảo số nguyên chính xác. Đối với đề thi, với địa chỉ ảo 21 (010101), và việc che dấu biến giá trị này thành 010000; sự thay đổi biến nó thành 01, hoặc trang ảo 1, như mong muốn. Sau đó, chúng tôi sử dụng giá trị này như một chỉ mục trong mảng các PTE đư ợc chỉ đến bởi thanh ghi cơ sở bảng trang.

Khi địa chỉ vật lý này đư ợc biết, phần cứng có thể lấy PTE từ bộ nhớ, trích xuất PFN và nối nó với phần bù từ địa chỉ ảo để tạo thành địa chỉ vật lý mong muốn. Cụ thể, bạn có thể nghĩ về PFN đư ợc SHIFT dịch sang trái, sau đó theo chiều kim loại HOẶC với phần bù để tạo thành địa chỉ cuối cùng như sau:

```
offset = VirtualAddress & OFFSET_MASK
PhysAddr = (PFN << SHIFT) | bù lại
```

Cuối cùng, phần cứng có thể lấy dữ liệu mong muốn từ bộ nhớ và đư a nó vào thanh ghi eax. Chư ơng trình hiện đã thành công khi tải một giá trị từ bộ nhớ!

Tóm lại, bây giờ chúng ta mô tả giao thức ban đầu cho những gì xảy ra trên mỗi tham chiếu bộ nhớ. Hình 18.6 (trang 9) cho thấy cách tiếp cận. Đối với mọi tham chiếu bộ nhớ (cho dù là tìm nạp lệnh hay tải hoặc lư u trữ rõ ràng), phân trang yêu cầu chúng ta thực hiện một tham chiếu bộ nhớ bổ sung để tìm nạp bản dịch từ bảng trang trư ớc. Đó là rất nhiều

```
1 // Giải nén VPN từ địa chỉ ảo
2 VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4 // Tạo địa chỉ của mục nhập bảng trang (PTE)
5 PTEAddr = PTBR + (VPN * sizeof (PTE))
6
7 // Tìm nạp PTE
8 PTE = AccessMemory (PTEAddr)
9
10 // Kiểm tra xem quá trình có thể truy cập trang 11 không nếu
(PTE.Valid == False)
12          RaiseException (SEGMENTATION_FAULT ) 13 else if
(CanAccess (PTE.ProtectBits) == Sai)
14          RaiseException (PROTECTION_FAULT)
15 ngư ời khác
16          // Truy cập là OK: tạo địa chỉ vật lý và tìm nạp nó offset = VirtualAddress &
17          OFFSET_MASK PhysAddr = (PTE.PFN << PFN_SHIFT) | bù đắp Đăng ký = AccessMemory
18          (PhysAddr)
19
```

Hình 18.6: Truy cập bộ nhớ bằng phân trang

công việc! Tham chiếu bộ nhớ bổ sung rất tốn kém và trong trư ờng hợp này có thể sẽ làm chậm quá trình đi một phần hai hoặc nhiều hơn.

Và bây giờ bạn có thể hy vọng rằng có hai vấn đề thực sự mà chúng ta phải giải quyết. Nếu không thiết kế cẩn thận cả phần cứng và phần mềm, bảng trang sẽ khiến hệ thống chạy quá chậm, cũng như chiếm quá nhiều bộ nhớ. Mặc dù dư ờng như là một giải pháp tuyệt vời cho nhu cầu ảo hóa bộ nhớ của chúng ta, như ng hai vấn đề quan trọng này trư ớc tiên phải đư ợc khắc phục.

18.5 Một dấu vết bộ nhớ

Trư ớc khi kết thúc, bây giờ chúng ta theo dõi thông qua một bài kiểm tra truy cập bộ nhớ đơn giản để chứng minh tất cả các kết quả truy cập bộ nhớ xảy ra khi sử dụng phân trang. Đoạn mã (trong C, trong tệp đư ợc gọi là array.c) mà chúng tôi quan tâm như sau:

```
int mảng [1000];
...
for (i = 0; i <1000; i ++) array [i] =
      0;
```

Chúng tôi biên dịch array.c và chạy nó bằng các lệnh sau:

```
nhắc> gcc -o mảng array.c -Gọi -O nhắc> ./array
```

Tất nhiên, để thực sự hiểu bộ nhớ mà con vật cưng snip mã này (chỉ đơn giản là khởi tạo một mảng) sẽ tạo ra những gì, chúng ta sẽ phải biết (hoặc giả sử) một vài điều nữa. Đầu tiên, chúng ta sẽ phải tháo rời kết quả trong tệp nhị phân (sử dụng objdump trên Linux hoặc otool trên Mac) để xem những hư ớng dẫn hợp ngữ nào đư ợc sử dụng để khởi tạo mảng trong một vòng lặp. Đây là mã lắp ráp kết quả:

```
1024 tháng $ 0x0, (% edi,% eax, 4) 1028
bao gồm% eax 1032 cmpl $ 0x03e8,% eax
1036 jne 0x1024
```

Mã, nếu bạn biết một chút x86, thực sự khá dễ hiểu[2] .
Lệnh đầu tiên di chuyển giá trị 0 (đư ợc hiển thị dư ới dạng $ 0x0) vào địa chỉ bộ nhớ vir của vị trí của mảng; địa chỉ này đư ợc tính bằng cách lấy nội dung của % edi và thêm % eax nhân với bốn vào nó.
Do đó, % edi giữ địa chỉ cơ sở của mảng, trong khi % eax giữ chỉ số mảng (i); chúng ta nhân với bốn vì mảng là một mảng gồm các ger nguyên, mỗi ký tự có kích thư ớc là bốn byte.

Lệnh thứ hai tăng chỉ số mảng đư ợc giữ bằng % eax và lệnh thứ ba so sánh nội dung của thanh ghi đó với giá trị hex 0x03e8 hoặc thập phân 1000. Nếu so sánh cho thấy hai giá trị chư a bằng nhau (đó là giá trị jne kiểm tra hư ớng dẫn), lệnh thứ tư nhảy trở lại đầu vòng lặp.

Để hiểu bộ nhớ nào truy cập chuỗi lệnh này (ở cả cấp độ ảo và vật lý), chúng ta sẽ phải giả định điều gì đó về vị trí trong bộ nhớ ảo mà đoạn mã và mảng đư ợc tìm thấy, cũng như nội dung và vị trí của bảng trang. .

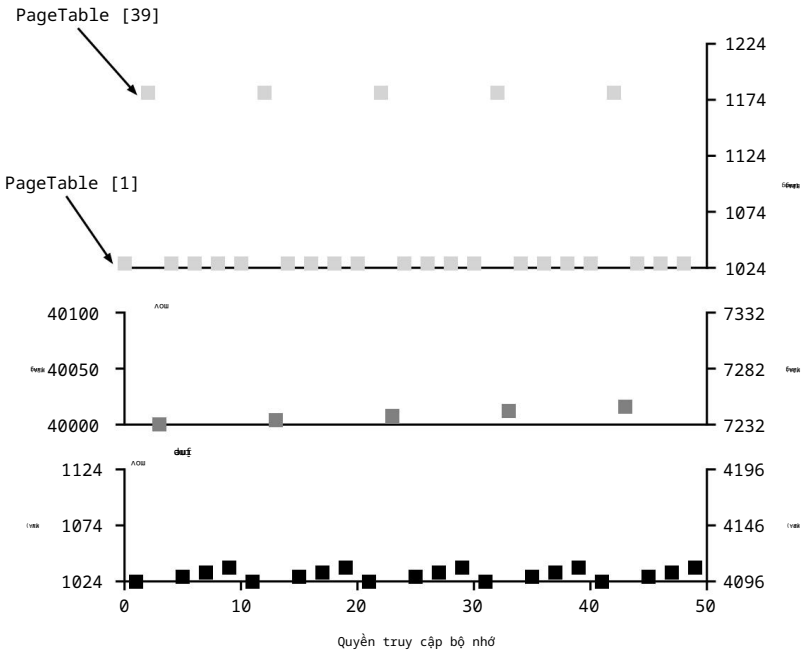
Đối với ví dụ này, chúng tôi giả định một không gian địa chỉ ảo có kích thư ớc 64KB (không thực tế là nhỏ). Chúng tôi cũng giả định kích thư ớc trang là 1KB.

Tất cả những gì chúng ta cần biết bây giờ là nội dung của bảng trang và vị trí của nó trong bộ nhớ vật lý. Giả sử chúng ta có một bảng trang tuyến tính (dựa trên mảng) và nó nằm ở địa chỉ vật lý 1KB (1024).

Đối với nội dung của nó, chỉ có một vài trang ảo mà chúng ta cần phải lo lắng về việc đã lập bản đồ cho ví dụ này. Đầu tiên, có trang ảo mà mã sử dụng. Vì kích thư ớc trang là 1KB nên địa chỉ ảo 1024 nằm trên trang thứ hai của không gian địa chỉ ảo (VPN = 1, vì VPN = 0 là trang đầu tiên). Giả sử trang ảo này ánh xạ đến khung vật lý 4 (VPN 1 PFN 4).

Tiếp theo, có mảng chính nó. Kích thư ớc của nó là 4000 byte (1000 số nguyên) và chúng tôi giả định rằng nó nằm ở các địa chỉ ảo 40000 đến 44000 (không bao gồm byte cuối cùng). Các trang ảo cho phạm vi thập phân này là VPN = 39 ... VPN = 42. Vì vậy, chúng tôi cần ánh xạ cho các trang này. Hãy giống như các ánh xạ ảo-vật lý này cho ví dụ: (VPN 39 PFN 7), (VPN 40 PFN 8), (VPN 41 PFN 9), (VPN 42 PFN 10).

[2] Chúng tôi đang gian lận một chút ở đây, giả sử mỗi lệnh có kích thư ớc bốn byte cho độ dẻo của sim; trên thực tế, các lệnh x86 có kích thư ớc thay đổi.



Hình 18.7: Dấu vết bộ nhớ ảo (và vật lý)

Bây giờ chúng tôi đã sẵn sàng để theo dõi các tham chiếu bộ nhớ của chương trình. Khi nó chạy, mỗi lần tìm nạp lệnh sẽ tạo ra hai tham chiếu bộ nhớ: một tham chiếu đến bảng trang để tìm khung vật lý mà lệnh nằm bên trong và một tham chiếu tới chính lệnh để tìm nạp nó vào CPU cho quá trình nhập. Ngoài ra, có một tham chiếu bộ nhớ rõ ràng ở dạng lệnh mov ; điều này thêm một quyền truy cập bảng trang khác trước tiên (để dịch địa chỉ ảo mảng sang địa chỉ vật lý chính xác) và sau đó là chính quyền truy cập mảng.

Toàn bộ quá trình, trong năm lần lặp vòng lặp đầu tiên, được mô tả trong Hình 18.7 (trang 11). Đồ thị phía dưới cùng hiển thị các tham chiếu bộ nhớ lệnh trên trục y màu đen (với địa chỉ ảo ở bên trái và địa chỉ vật lý thực tế ở bên phải); biểu đồ giữa hiển thị các truy cập mảng có màu xám đậm (một lần nữa với ảo ở bên trái và vật lý ở bên phải); Cuối cùng, biểu đồ trên cùng hiển thị các truy cập bộ nhớ bảng trang có màu xám nhạt (chỉ là vật lý, vì bảng trang trong ví dụ này nằm trong ory mem vật lý). Trục x, đối với toàn bộ dấu vết, hiển thị các truy cập bộ nhớ trong năm lần lặp đầu tiên của vòng lặp; có 10 lần truy cập bộ nhớ trên mỗi vòng lặp, bao gồm bốn lần tìm nạp lệnh, một lần cập nhật bộ nhớ rõ ràng và năm lần truy cập bảng trang để dịch bốn lần tìm nạp đó và một lần cập nhật rõ ràng.

Hãy xem liệu bạn có thể hiểu được các mẫu hiển thị trong quá trình hiển thị trực quan hóa này hay không. Đặc biệt, điều gì sẽ thay đổi khi vòng lặp tiếp tục chạy ngoài năm lần lặp đầu tiên này? Vị trí bộ nhớ mới nào sẽ là đã truy cập? Bạn có thể hình dung về nó?

Đây chỉ là ví dụ đơn giản nhất (chỉ một vài dòng mã C), và bạn vẫn có thể cảm nhận được sự phức tạp của việc hiểu hành vi bộ nhớ thực tế của các ứng dụng thực. Đừng lo lắng: nó chắc chắn sẽ trở nên tồi tệ hơn, bởi vì các cơ chế mà chúng tôi sắp giới thiệu chỉ làm phức tạp bộ máy vốn đã phức tạp này. Xin lỗi3 !

18.6 Tóm tắt

Chúng tôi đã giới thiệu khái niệm phân trang như một giải pháp cho khả năng ảo hóa bộ nhớ của chúng tôi. Phân trang có nhiều lợi thế hơn so với các phương pháp tiếp cận previous (chẳng hạn như phân đoạn). Đầu tiên, nó không dẫn đến bên ngoài phân mảnh, vì phân trang (theo thiết kế) chia bộ nhớ thành kích thước cố định các đơn vị. Thứ hai, nó khá linh hoạt, cho phép sử dụng thưa thớt các không gian quảng cáo ảo.

Tuy nhiên, việc thực hiện hỗ trợ phân trang mà không cẩn thận sẽ dẫn đến máy chậm hơn (với nhiều bộ nhớ phụ truy cập để truy cập trang bảng) cũng như lãng phí bộ nhớ (với bộ nhớ chứa đầy các bảng trang thay vì dữ liệu ứng dụng hữu ích). Do đó, chúng tôi sẽ phải suy nghĩ kỹ hơn một chút để đưa ra một hệ thống phân trang không chỉ hoạt động mà còn hoạt động tốt. May mắn thay, hai chương tiếp theo sẽ chỉ cho chúng ta cách làm như vậy.

3Chúng tôi không thực sự xin lỗi. Nhưng, chúng tôi xin lỗi về việc không xin lỗi, nếu điều đó có ý nghĩa.

Ngư ời giới thiệu

[KE + 62] “Hệ thống lư u trữ một cấp” của T. Kilburn, DBG Edwards, MJ Lanigan, FH Sum ner. IRE Trans. EC-11, 2, 1962. Đư ợc in lại trong Bell và Newell, “Cấu trúc máy tính: Bài đọc và ví dụ”. McGraw-Hill, New York, 1971. Atlas đi tiên phong trong ý tư ởng chia bộ nhớ thành các trang có kích thư ớc cố định và theo nhiều nghĩa là hình thức ban đầu của các ý tư ởng quản lý bộ nhớ mà chúng ta thấy trong các hệ thống máy tính hiện đại.

[I09] “Sách hư ớng dẫn của nhà phát triển phần mềm kiến trúc Intel 64 và IA-32” Intel, 2009. Có sẵn: http://www.intel.com/products/processor/manuals. Đặc biệt, hãy chú ý đến “Tập 3A: Hư ớng dẫn Lập trình Hệ thống Phần 1” và “Tập 3B: Hư ớng dẫn Lập trình Hệ thống Phần 2”.

[L78] “Manchester Mark I và Atlas: Một viễn cảnh lịch sử” của SH Lavington. Thông báo về ACM, Tập 21: 1, tháng 1 năm 1978. Bài báo này là một hồi tư ởng tuyệt vời về một số lịch sử phát triển của một số hệ thống máy tính quan trọng. Đôi khi chúng ta quên ở Mỹ, nhiều ý tư ởng mới này đến từ nư ớc ngoài.

Bài tập về nhà (Mô phỏng)

Trong bài tập về nhà này, bạn sẽ sử dụng một chương trình đơn giản, được gọi là paging-linear-translate.py, để xem liệu bạn có hiểu cách dịch địa chỉ ảo sang vật lý đơn giản hoạt động như thế nào với các bảng trang tuyến tính hay không. Xem README để biết chi tiết.

Câu hỏi 1.

Trước khi thực hiện bất kỳ bản dịch nào, hãy sử dụng trình mô phỏng để nghiên cứu cách các bảng trang tuyến tính thay đổi kích thước với các tham số khác nhau. Tính toán kích thước của bảng trang tuyến tính khi các thông số khác nhau thay đổi. Dưới đây là một số đầu vào gợi ý; bằng cách sử dụng cờ -v, bạn có thể xem có bao nhiêu mục nhập bảng trang được lấp đầy. Trước tiên, để hiểu kích thước bảng trang tuyến tính thay đổi như thế nào khi không gian địa chỉ tăng lên, hãy chạy với các cờ sau:

```
-P 1k -a 1m -p 512m -v -n 0
-P 1k -a 2m -p 512m -v -n 0
-P 1k -a 4m -p 512m -v -n 0
```

Sau đó, để hiểu kích thước bảng trang tuyến tính thay đổi như thế nào khi kích thước trang tăng lên:

```
-P 1k -a 1m -p 512m -v -n 0
-P 2k -a 1m -p 512m -v -n 0
-P 4k -a 1m -p 512m -v -n 0
```

Trước khi chạy bất kỳ cái nào trong số này, hãy thử nghĩ về các xu hướng dự kiến. Kích thước bảng trang sẽ thay đổi như thế nào khi không gian địa chỉ tăng lên? Khi kích thước trang tăng lên? Tại sao không sử dụng các trang lớn nói chung?

2. Bây giờ chúng ta hãy thực hiện một số bản dịch. Bắt đầu với một số ví dụ nhỏ và thay đổi số lượng trang được phân bổ cho không gian địa chỉ bằng cờ -u. Ví dụ:

```
-P 1k -a 16k -p 32k -v -u 0
-P 1k -a 16k -p 32k -v -u 25
-P 1k -a 16k -p 32k -v -u 50
-P 1k -a 16k -p 32k -v -u 75
-P 1k -a 16k -p 32k -v -u 100
```

Điều gì xảy ra khi bạn tăng tỷ lệ số trang được đặt trong mỗi không gian địa chỉ?

3. Bây giờ chúng ta hãy thử một số hạt giống ngẫu nhiên khác nhau và một số tham số không gian địa chỉ khác nhau (và đôi khi khá điên rồ), để đa dạng:

```
-P 8 -a 32 -p 1024 -v -s 1
-P 8k -a 32k -p 1m -v -s 2
-P 1m -a 256m -p 512m -v -s 3
```

Kết hợp tham số nào trong số những kết hợp tham số này là không thực tế? Tại sao?

4. Sử dụng chư ơng trình để thử một số vấn đề khác. Bạn có thể tìm thấy giới hạn của nơi chư ơng trình không hoạt động nữa không? Ví dụ: điều gì xảy ra nếu kích thư ớc không gian địa chỉ lớn hơn ory me vật lý?