QUẢN LÝ BỘ NHỚ

I Mục đích

Sau khi học xong chương này, người học nắm được những kiến thức sau:

- Hiểu các cách khác nhau để quản lý bộ nhớ
- Hiểu tiếp cận quản lý bộ phân trang và phân đoạn
- Vận dụng một tiếp cận quản lý bộ nhớ phù hợp với hệ thống xác định

II Giới thiệu

Trong chương này chúng ta sẽ thảo luận nhiều cách khác nhau để quản lý bộ nhớ. Các giải thuật quản lý bộ nhớ từ tiếp cận máy trơ cơ bản (primitive bare-machine) là chiến lược phân trang và phân đoạn. Mỗi tiếp cận có lợi điểm và nhược của chính nó. Chọn phương pháp quản lý bộ nhớ cho một hệ thống xác định phụ thuộc vào nhiều yếu tố, đặc biệt trên thiết kế phần cứng của hệ thống. Chúng ta sẽ thấy nhiều giải thuật yêu cầu hỗ trợ phần cứng mặc dù các thiết kế gần đây đã tích hợp phần cứng và hệ điều hành.

III Đặt vấn đề

Bộ nhớ là trung tâm để điều hành hệ thống máy tính hiện đại. Bộ nhớ chứa một mảng lớn các từ (words) hay các bytes, mỗi phần tử với địa chỉ của chính nó. CPU lấy các chỉ thị từ bộ nhớ dựa theo giá trị của thanh đếm chương trình. Các chỉ thị này có thể gây việc nạp bổ sung các từ và lưu trữ tới các địa chỉ bộ nhớ xác định.

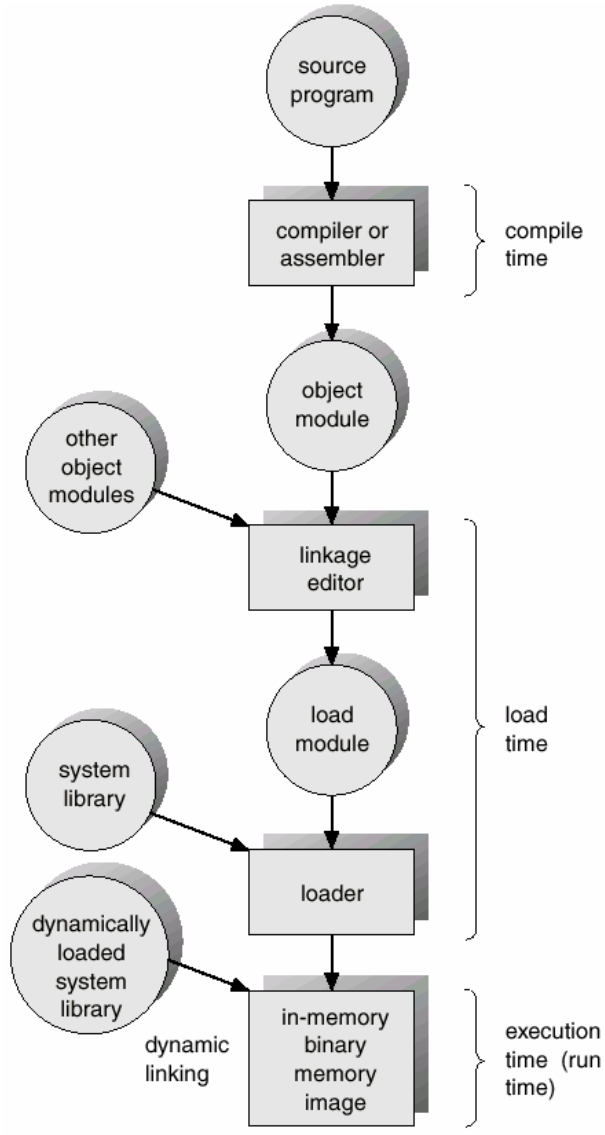
III.1 Liên kết địa chỉ

Thông thường, một chương trình nằm trên đĩa như một tập tin có thể thực thi dạng nhị phân. Chương trình này được mang vào trong bộ nhớ và được đặt trong một quá trình để nó được thực thi. Phụ thuộc vào việc quản lý bộ nhớ đang dùng, quá trình có thể được di chuyển giữa đĩa và bộ nhớ trong khi thực thi. Tập hợp các quá trình trên đĩa đang chờ được mang vào bộ nhớ để thực thi hình thành một **hàng đợi nhập** (input queue).

Thủ tục thông thường là chọn một trong những quá trình trong hàng đợi nhập và nạp quá trình đó vào trong bộ nhớ. Khi một quá trình được thực thi, nó truy xuất các chỉ thị và dữ liệu từ bộ nhớ. Cuối cùng, một quá trình kết thúc và không gian bộ nhớ của nó được xác định là trống.

Hầu hết các hệ thống cho phép một quá trình người dùng nằm ở bất cứ phần nào của bộ nhớ vật lý. Do đó, mặc dù không gian địa chỉ của máy tính bắt đầu tại 00000, nhưng địa chỉ đầu tiên của quá trình người dùng không cần tại 00000. Sắp xếp này ảnh hưởng đến địa chỉ mà chương trình người dùng có thể dùng. Trong hầu hết các trường hợp, một chương trình người dùng sẽ đi qua một số bước- một vài trong chúng có thể là tuỳ chọn-trước khi được thực thi (hình VII-1). Các địa chỉ có thể được hiện diện trong những cách khác trong những bước này. Các địa chỉ trong chương trình nguồn thường là những danh biểu. Một trình biên dịch sẽ liên kết các địa chỉ danh biểu tới các địa chỉ có thể tái định vị (chẳng hạn như 14 bytes từ vị trí bắt đầu của

module này). Bộ soạn thảo liên kết hay bộ nạp sẽ liên kết các địa chỉ có thể tái định vị tới địa chỉ tuyệt đối (chẳng hạn 74014). Mỗi liên kết là một ánh xạ từ một không gian địa chỉ này tới một không gian địa chỉ khác



Hình 0-1 Xử lý nhiều bước của chương trình người dùng

Về truyền thống, liên kết các chỉ thị và dữ liệu tới các địa chỉ có thể được thực hiện tại bất cứ bước nào theo cách sau đây:

- **Thời gian biên dịch**: nếu tại thời điểm biên dịch có thể biết quá trình nằm ở đâu trong bộ nhớ thì mã tuyệt đối có thể được phát sinh. Thí dụ, nếu biết trước quá trình người dùng nằm tại vị trí R thì mã trình biên dịch được phát sinh sẽ bắt đầu tại vị trí đó và mở rộng từ đó. Nếu tại thời điểm sau đó, vị trí bắt đầu thay đổi thì sẽ cần biên dịch lại mã này. Các chương trình định dạng .COM của MS-DOS là mã tuyệt đối giới hạn tại thời điểm biên dịch.
- **Thời điểm nạp**: nếu tại thời điểm biên dịch chưa biết nơi quá trình sẽ nằm ở đâu trong bộ nhớ thì trình biên dịch phải phát sinh mã có thể tái định vị. Trong trường hợp này, liên kết cuối cùng được trì hoãn cho tới thời điểm

nạp. Nếu địa chỉ bắt đầu thay đổi, chúng ta chỉ cần nạp lại mã người dùng để hợp nhất giá trị được thay đổi này.

- **Thời gian thực thi**: nếu quá trình có thể được di chuyển trong thời gian thực thi từ một phân đoạn bộ nhớ này tới một phân đoạn bộ nhớ khác thì việc liên kết phải bị trì hoãn cho tới thời gian chạy. Phần cứng đặc biệt phải sẵn dùng cho cơ chế này để thực hiện công việc. Hầu hết những hệ điều hành này dùng phương pháp này.

Phần chủ yếu của chương này được dành hết để hiển thị các liên kết khác nhau có thể được cài đặt hữu hiệu trong một hệ thống máy tính và thảo luận sự hỗ trợ phần cứng tương ứng.

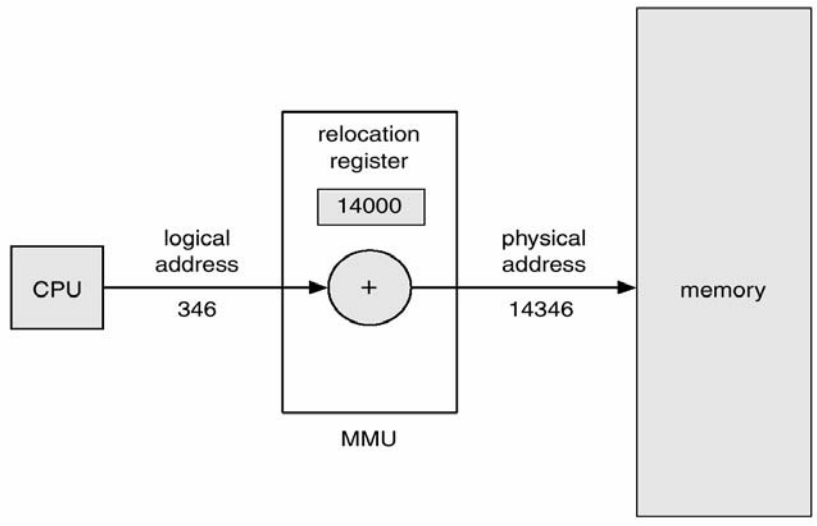
III.2 Không gian địa chỉ luận lý và không gian địa chỉ vật lý

Một địa chỉ được tạo ra bởi CPU thường được gọi là **địa chỉ luận lý** (logical address), ngược lại một địa chỉ được xem bởi đơn vị bộ nhớ-nghĩa là, một địa chỉ được nạp vào thanh ghi địa chỉ bộ nhớ-thường được gọi là **địa chỉ vật lý** (physical address).

Các phương pháp liên kết địa chỉ thời điểm biên dịch và thời điểm nạp tạo ra địa chỉ luận lý và địa chỉ vật lý xác định. Tuy nhiên, cơ chế liên kết địa chỉ tại thời điểm thực thi dẫn đến sự khác nhau giữa địa chỉ luận lý và địa chỉ vật lý. Trong trường hợp này, chúng ta thường gọi địa chỉ luận lý như là **địa chỉ ảo** (virtual address). Tập hợp tất cả địa chỉ luận lý được tạo ra bởi chương trình là **không gian địa chỉ luận lý** ; tập hợp tất cả địa chỉ vật lý tương ứng địa chỉ luận lý này là **không gian địa chỉ vật lý**. Do đó, trong cơ chế liên kết địa chỉ tại thời điểm thực thi, không gian địa chỉ luận lý và không gian địa chỉ vật lý là khác nhau.

Việc ánh xạ tại thời điểm thực thi từ địa chỉ ảo tới địa chỉ vật lý được thực hiện bởi một thiết bị phần cứng được gọi là **bộ quản lý bộ nhớ MMU** (memory-management unit). Chúng ta có thể chọn giữa những phương pháp khác nhau để thực hiện việc ánh xạ.

Như được hiển thị trong hình VII-2 ở trên, phương pháp này yêu cầu sự hỗ trợ phần cứng. Thanh ghi nền bây giờ được gọi là **thanh ghi tái định vị**. Giá trị trong thanh ghi tái định vị được cộng vào mỗi địa chỉ được tạo ra bởi quá trình người dùng tại thời điểm nó được gởi tới bộ nhớ. Thí dụ, nếu giá trị nền là 14000, thì việc cố gắng bởi người dùng để xác định vị trí 0 được tự động tái định vị tới vị trí 14000; một truy xuất tới địa chỉ 346 được ánh xạ tới vị trí 14346.



Hình 0-2 định vị tự động dùng thanh ghi tái định vị

III.3 Nạp động

Trong thảo luận của chúng ta gần đây, toàn bộ chương trình và dữ liệu của một quá trình phải ở trong bộ nhớ vật lý để quá trình thực thi. Kích thước của quá trình bị giới hạn bởi kích thước của bộ nhớ vật lý. Để đạt được việc sử dụng không gian bộ nhớ tốt hơn, chúng ta có thể sử dụng **nạp động** (dynamic loading). Với nạp động, một thủ tục không được nạp cho tới khi nó được gọi. Tất cả thủ tục được giữ trên đĩa trong định dạng nạp có thể tái định vị. Chương trình chính được nạp vào bộ nhớ và được thực thi. Khi một thủ tục cần gọi một thủ tục khác, thủ tục gọi trước hết kiểm tra để thấy thủ tục khác được nạp hay không. Nếu không, bộ nạp liên kết có thể tái định vị được gọi để nạp thủ tục mong muốn vào bộ nhớ và cập nhật các bảng địa chỉ của chương trình để phản ánh thay đổi này. Sau đó, điều khiển này được truyền tới thủ tục mới được nạp.

Thuận lợi của nạp động là ở đó một thủ tục không được dùng thì không bao giờ được nạp. Phương pháp này đặc biệt có ích khi lượng lớn mã được yêu cầu quản lý các trường hợp xảy ra không thường xuyên, chẳng hạn như các thủ tục lỗi. Trong trường hợp này, mặc dù kích thước toàn bộ chương trình có thể lớn, nhưng phần được dùng (và do đó được nạp) có thể nhỏ hơn nhiều.

Nạp động không yêu cầu hỗ trợ đặc biệt từ hệ điều hành. Nhiệm vụ của người dùng là thiết kế các chương trình của họ để đạt được sự thuận lợi đó. Tuy nhiên, hệ điều hành có thể giúp người lập trình bằng cách cung cấp các thủ tục thư viện để cài đặt nạp tự động.

III.4 Liên kết động và các thư viện được chia sẻ

Trong hình VII-1 cũng hiển thị thư viện được liên kết động. Một số hệ điều hành hỗ trợ chỉ liên kết tĩnh mà trong đó các thư viện ngôn ngữ hệ thống được đối xử như bất kỳ module đối tượng khác và được kết hợp bởi bộ nạp thành hình ảnh chương trình nhị phân. Khái niệm liên kết động là tương tự như khái niệm nạp động. Liên kết bị trì hoãn hơn là việc nạp bị trì hoãn cho tới thời điểm thực thi. Đặc điểm này thường được dùng với các thư viện hệ thống như các thư viện chương trình con của các ngôn ngữ. Không có tiện ích này, tất cả chương trình trên một hệ thống cần có một bản sao thư viện của ngôn ngữ của chúng (hay ít nhất thư viện được tham chiếu bởi chương trình) được chứa trong hình ảnh có thể thực thi. Yêu cầu này làm lãng phí cả không gian đĩa và bộ nhớ chính. Với liên kết động, một **stub** là một đoạn mã hiển thị cách định vị chương trình con trong thư viện cư trú trong bộ nhớ hay cách nạp thư viện nếu chương trình con chưa hiện diện.

Khi stub này được thực thi, nó kiểm tra để thấy chương trình con được yêu cầu đã ở trong bộ nhớ hay chưa. Nếu chưa, chương trình này sẽ nạp chương trình con vào trong bộ nhớ. Dù là cách nào, stub thay thế chính nó với địa chỉ của chương trình con và thực thi chương trình con đó. Do đó, thời điểm tiếp theo phân đoạn mã đạt được, chương trình con trong thư viện được thực thi trực tiếp mà không gây ra bất kỳ chi phí cho việc liên kết động. Dưới cơ chế này, tất cả các quá trình sử dụng một thư viện ngôn ngữ thực thi chỉ một bản sao của mã thư viện.

III.5 Phủ lắp

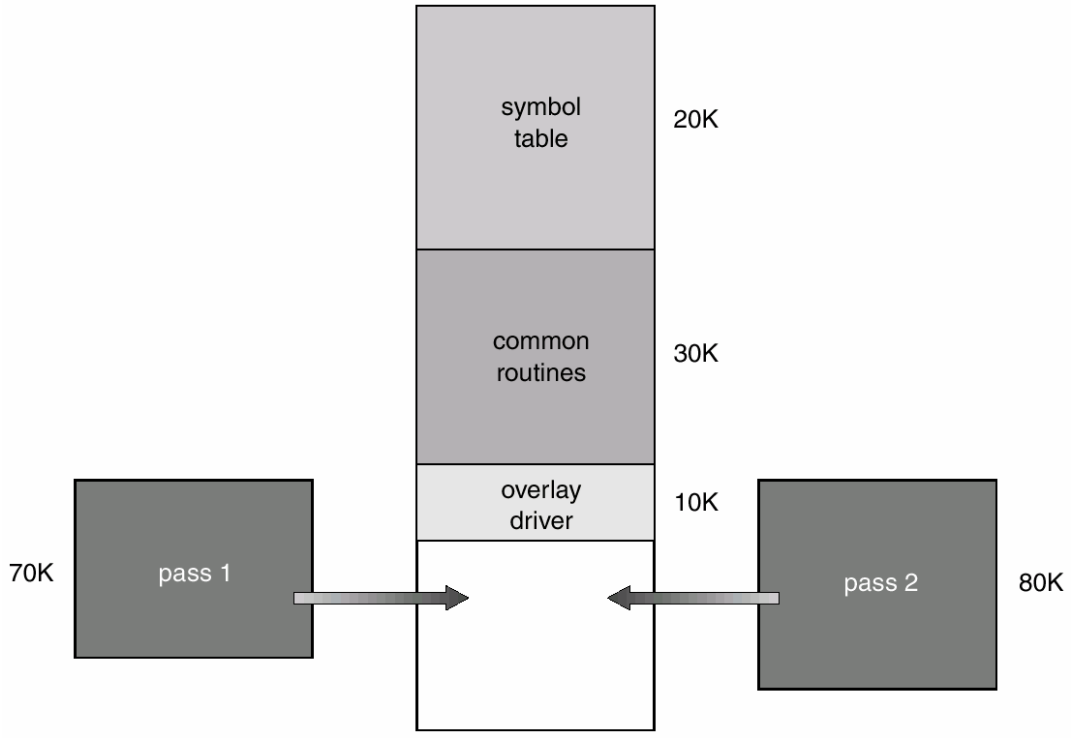
Để cho phép một quá trình lớn hơn lượng bộ nhớ được cấp phát cho nó, chúng ta sử dụng **cơ chế phủ lắp** (overlays). Ý tưởng phủ lắp là giữ trong bộ nhớ những chỉ thị và dữ liệu được yêu cầu tại bất kỳ thời điểm nào được cho. Khi những chỉ thị đó được yêu cầu, chúng được nạp vào không gian được chiếm trước đó bởi các chỉ thị mà chúng không còn cần nữa.

Thí dụ, xét trình dịch hợp ngữ hai lần (two-pass assembler). Trong suốt lần thứ 1, nó xây dựng bảng danh biểu; sau đó, trong lần thứ 2, nó tạo ra mã máy. Chúng ta có thể phân chia trình dịch hợp ngữ thành mã lần 1, mã lần 2, bảng danh biểu, và những thủ tục hỗ trợ chung được dùng bởi lần 1 và lần 2. Giả sử kích thước của các thành phần này như sau:

Lần 1	70 KB
Lần 2	80 KB
Bảng danh biểu	20 KB
Các thủ tục chung	30 KB

Để nạp mọi thứ một lần, chúng ta cần 200KB bộ nhớ. Nếu chỉ có 150KB sẵn có, chúng ta không thể chạy quá trình của chúng ta. Tuy nhiên, chú ý rằng lần 1 và lần 2 không cần ở trong bộ nhớ cùng một lúc. Do đó, chúng ta định nghĩa hai phủ lắp. Phủ lắp A là bảng danh biểu, các thủ tục chung, lần 1, và phủ lắp B là bảng biểu tượng, các thủ tục chung và lần 2.

Chúng ta bổ sung trình điều khiển phủ lắp (10 KB) và bắt đầu với phủ lắp A trong bộ nhớ. Khi chúng ta kết thúc lần 1, chúng ta nhảy tới trình điều khiển phủ lắp, trình điều khiển này sẽ đọc phủ lắp B vào trong bộ nhớ, viết chồng lên phủ lắp B và sau đó chuyển điều khiển tới lần 2. Phủ lắp A chỉ cần 120KB, ngược lại phủ lắp B cần 130KB (hình VII-3). Bây giờ chúng ta có thể chạy trình hợp ngữ trong 150KB bộ nhớ. Nó sẽ nạp nhanh hơn vì rất ít dữ liệu cần được chuyển trước khi việc thực thi bắt đầu. Tuy nhiên, nó sẽ chạy chậm hơn do nhập/xuất phụ đọc mã mã cho phủ lắp A qua mã cho phủ lắp B.

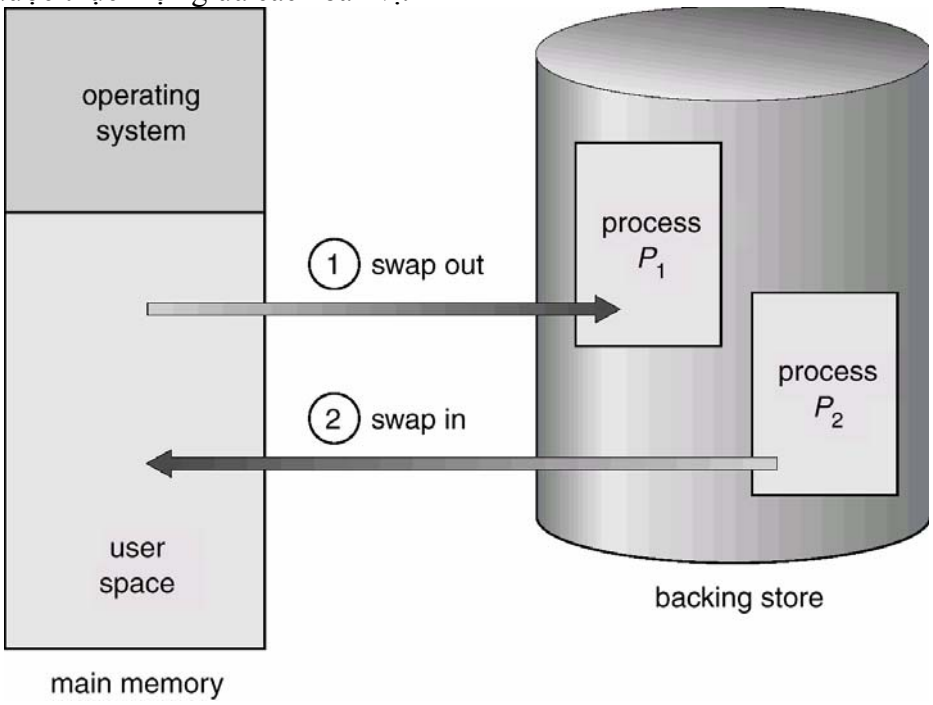


Hình 0-3- Các phủ lắp cho một bộ hợp ngữ dịch hai lần

Mã cho phủ lắp A và mã cho phủ lắp B được giữ trên đĩa như những hình ảnh bộ nhớ tuyệt đối, và được đọc bởi trình điều khiển phủ lắp khi cần. Tái định vị đặc biệt và các giải thuật liên kết được yêu cầu xây dựng các phủ lắp.

IV Hoán vị

Một quá trình cần ở trong bộ nhớ để được thực thi. Tuy nhiên, một quá trình có thể được hoán vị (swapped) tạm thời khỏi bộ nhớ tới vùng lưu trữ phụ ***backing store***, sau đó mang trở lại bộ nhớ để việc thực thi được tiếp tục. Thí dụ, giả sử một môi trường đa chương với giải thuật lập thời biểu CPU round-robin. Khi định mức thời gian hết, bộ quản lý bộ nhớ sẽ bắt đầu **hoán vị ra** (swap out) vùng lưu trữ phụ quá trình vừa mới kết thúc và **hoán vị vào** (swap in) một quá trình khác tới không gian bộ nhớ được trống (hình VII-4). Do đó, bộ định thời biểu CPU sẽ cấp những phần thời gian tới những quá trình khác trong bộ nhớ. Lý tưởng, bộ quản lý sẽ hoán vị các quá trình đủ nhanh để một vài quá trình sẽ ở trong bộ nhớ, sẵn sàng thực thi, khi bộ định thời CPU muốn định thời lại CPU. Định mức cũng phải đủ lớn để phù hợp lượng tính toán được thực hiện giữa các hoán vị.



Hình 0-4- Hoán vị hai quá trình dùng đĩa như là backing store

Một biến thể của chính sách hoán vị này được dùng cho các giải thuật định thời trên cơ sở ưu tiên. Nếu một quá trình có độ ưu tiên cao hơn đến và muốn phụ vụ, bộ quản lý bộ nhớ có thể hoán vị ra quá trình có độ ưu tiên thấp hơn để mà nó có thể nạp và thực thi quá trình có độ ưu tiên cao hơn. Khi quá trình có độ ưu tiên cao hơn kết thúc, quá trình có độ ưu tiên thấp hơn có thể được hoán vị vào trở lại và được tiếp tục. Biến thể của hoán vị này thường được gọi là **cuộn ra** (roll out), và **cuộn vào** (roll in).

Thông thường, một quá trình được hoán vị ra sẽ được hoán vị trở lại vào cùng không gian bộ nhớ mà nó đã chiếm trước đó. Sự giới hạn này được sai khiến bởi phương pháp liên kết địa chỉ. Nếu liên kết địa chỉ được thực hiện tại thời điểm hợp dịch hay nạp thì quá trình không thể được di chuyển vào không gian bộ nhớ khác vì các địa chỉ vật lý được tính trong thời gian thực thi.

Hoán vị yêu cầu một **vùng lưu trữ phụ** (backing store). Vùng lưu trữ phụ này thường là một đĩa tốc độ cao. Nó phải đủ lớn để chứa các bản sao của tất cả hình ảnh bộ nhớ cho tất cả người dùng, và nó phải cung cấp truy xuất trực tiếp tới các hình ảnh bộ nhớ này. Hệ thống này duy trì một **hàng đợi sẵn sàng** chứa tất cả quá trình mà các hình ảnh bộ nhớ của nó ở trong vùng lưu trữ phụ hay trong bộ nhớ và sẵn sàng để thực thi. Bất cứ khi nào bộ định thời CPU quyết định thực thi một quá trình, nó gọi **bộ phân phát** (dispacher). Bộ phân phát kiểm tra để thấy quá trình tiếp theo trong hàng đợi ở trong bộ nhớ không. Nếu không, và không có vùng bộ nhớ trống, bộ phân phát hoán vị ra một quá trình hiện hành trong bộ nhớ và hoán vị vào một quá trình mong muốn. Sau đó, nó nạp lại các thanh ghi và chuyển điều khiển tới quá trình được chọn. Trong các hệ hoán vị, thời gian chuyển đổi giữa các tác vụ cần được quan tâm. Mỗi quá trình cần được phân chia một khoảng thời gian sử dụng CPU đủ lớn để không thấy rõ sự chậm trễ do các thao tác hoán vị gây ra. Nếu không, hệ thống sẽ dùng phần lớn thời gian để hoán vị các quá trình vào ra bộ nhớ chính, CPU như vậy sẽ không sử dụng hiệu quả.

Hoán vị cũng bị ràng buộc bởi yếu tố khác. Nếu chúng ta muốn hoán vị một quá trình, chúng ta phải đảm bảo rằng nó hoàn toàn rỗi. Quan tâm đặc biệt là việc chờ nhập/xuất. Một quá trình có thể đang chờ thao tác nhập/xuất khi chúng ta hoán vị quá trình đó tới nơi trống bộ nhớ của nó. Tuy nhiên, nếu nhập/xuất đang truy xuất không đồng bộ bộ nhớ người dùng cho nhập/xuất vùng đệm, thì quá trình không thể được hoán vị. Giả sử rằng thao tác nhập/xuất đang xếp hàng chờ vì thiết bị đang bận. Sau đó, nếu chúng ta hoán vị quá trình P_1 ra và hoán vị P_2 vào thì thao tác nhập/xuất có thể cố gắng sử dụng bộ nhớ hiện thuộc về quá trình P_2. Hai giải pháp chủ yếu cho quá trình này là không bao giờ hoán vị quá trình đang chờ nhập/xuất hay thực thi các thao tác nhập/xuất chỉ ở trong vùng đệm của hệ điều hành. Chuyển giữa các vùng đệm của hệ điều hành và bộ nhớ quá trình thì chỉ xảy ra khi quá trình được hoán vị vào.

V Cấp phát bộ nhớ liên tục

Bộ nhớ chính phải cung cấp cho cả hệ điều hành và các quá trình người dùng khác nhau. Do đó, chúng ta cần cấp phát những phần khác nhau của bộ nhớ chính trong những cách hiệu quả nhất có thể. Phần này chúng ta giải thích một phương pháp thông dụng, cấp phát bộ nhớ liên tục.

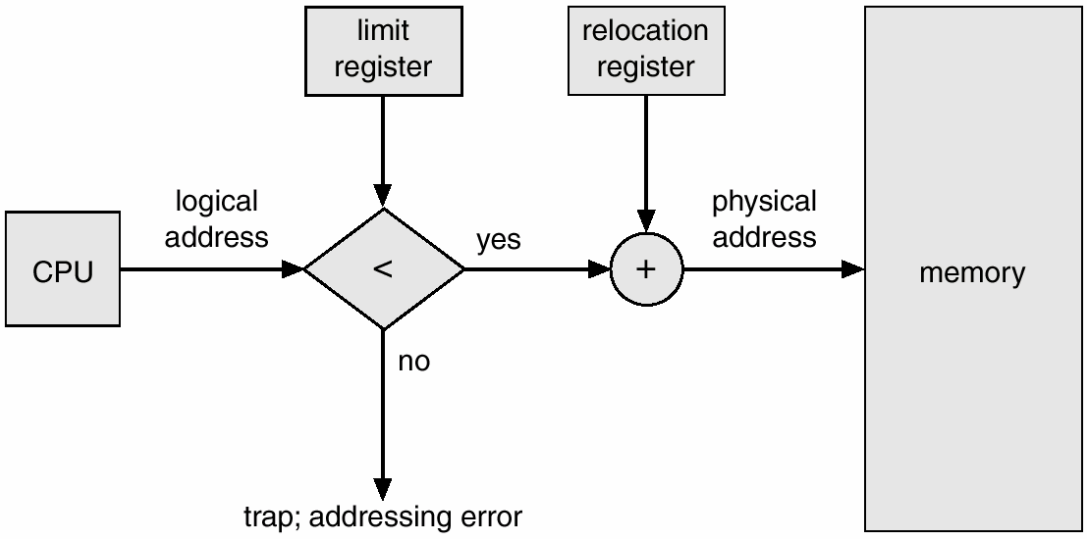
Bộ nhớ thường được phân chia thành hai phân khu, một cho hệ điều hành định vị và một cho các quá trình người dùng. Chúng ta có thể đặt hệ điều hành ở bộ nhớ cao hay bộ nhớ thấp. Yếu tố quan trọng ảnh hưởng tới quyết định này là vị trí của vector ngắt. Vì vector ngắt thường ở trong bộ nhớ thấp nên các lập trình viên thường cũng đặt hệ điều hành trong bộ nhớ thấp. Do đó, trong giáo trình này chúng ta sẽ thảo luận chỉ trường hợp hệ điều hành định vị trong bộ nhớ thấp. Phát triển của trường hợp khác là tương tự.

Chúng ta thường muốn nhiều quá trình người dùng định vị trong bộ nhớ tại cùng thời điểm. Do đó, chúng ta cần xem xét cách cấp phát bộ nhớ trống tới những quá trình ở trong hàng đợi nhập đang chờ được mang vào bộ nhớ. Trong cấp phát bộ nhớ liên tục, mỗi quá trình được chứa trong một phần bộ nhớ liên tục.

V.1 Bảo vệ bộ nhớ

Trước khi thảo luận cấp phát bộ nhớ chúng ta phải thảo luận vấn đề bảo vệ bộ nhớ-bảo vệ hệ điều hành từ quá trình người dùng, và bảo vệ các quá trình từ một quá trình khác. Chúng ta có thể cung cấp bảo vệ này bằng cách dùng thanh ghi tái định vị.

Thanh ghi tái định vị chứa giá trị địa chỉ vật lý nhỏ nhất; thanh ghi giới hạn chứa dãy các định chỉ luận lý (thí dụ: tái định vị = 100040 và giới hạn = 74600). Với các thanh ghi tái định vị và giới hạn, mỗi địa chỉ luận lý phải ít hơn thanh ghi giới hạn; MMU ánh xạ địa chỉ luận lý động bằng cách cộng giá trị trong thanh ghi tái định vị. Địa chỉ được tái định vị này được gửi tới bộ nhớ (như hình VII-5).



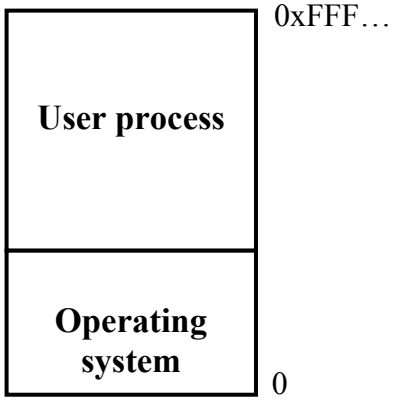
Hình 0-5 Hỗ trợ phần cứng cho các thanh ghi tái định vị và các giới hạn

Khi bộ định thời CPU chọn một quá trình thực thi, bộ phân phát nạp thanh ghi tái định vị và giới hạn với các giá trị đúng như một phần của chuyển đổi ngữ cảnh. Vì mọi địa chỉ được phát sinh bởi CPU được kiểm tra dựa trên các thanh ghi này, chúng ta có thể bảo vệ hệ điều hành và các chương trình và dữ liệu người dùng khác từ việc sửa đổi bởi quá trình đang chạy này.

Cơ chế dùng thanh ghi tái định vị cung cấp một cách hiệu quả để cho phép kích thước hệ điều hành thay đổi động. Khả năng mềm dẽo này có thể mong muốn trong nhiều trường hợp. Thí dụ, hệ điều hành chứa mã và không gian vùng đệm cho trình điều khiển thiết bị. Nếu một trình điều khiển thiết bị (hay dịch vụ hệ điều hành khác) không được dùng phổ biến, nó không muốn giữ mã và dữ liệu trong bộ nhớ, khi chúng ta có thể dùng không gian đó cho mục đích khác. Những mã như thế thường được gọi là mã hệ điều hành tạm thời (transient operating system code); nó đến và đi khi được yêu cầu. Do đó, dùng mã này thay đổi kích thước của hệ điều hành trong khi thực thi chương trình.

V.2 Hệ thống đơn chương

Trong phương pháp này bộ nhớ được chia sẻ cho hệ điều hành và một chương trình duy nhất của người sử dụng. Tại một thời điểm, một phần của bộ nhớ sẽ do hệ điều hành chiếm giữ, phần còn lại thuộc về quá trình người dùng duy nhất trong hệ thống (Hình VII-6). Quá trình này được toàn quyền sử dụng bộ nhớ dành cho nó.



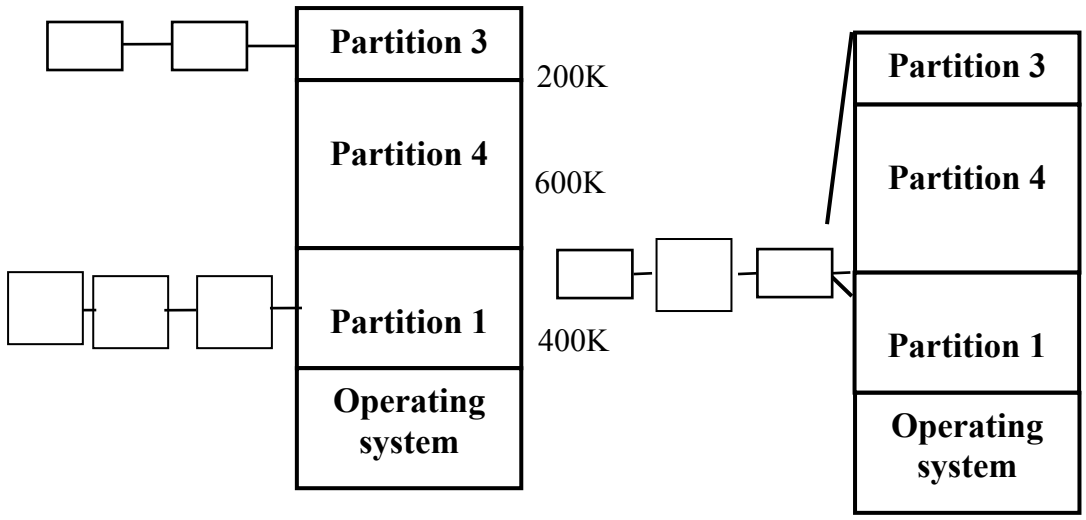
Hình 0-6 Tổ chức bộ nhớ trong hệ thống đơn chương

Khi bộ nhớ được tổ chức theo cách thức này, chỉ có thể xử lý một chương trình tại một thời điểm. Quan sát hoạt động của các quá trình, có thể nhận thấy rất nhiều tiến trình trải qua phần lớn thời gian để chờ các thao tác nhập/xuất hoàn thành. Trong suốt thời gian này, CPU ở trạng thái rỗi. Trong trường hợp như thế, hệ thống đơn chương không cho phép sử dụng hiệu quả CPU. Ngoài ra, sự đơn chương không cho phép nhiều người sử dụng làm việc đồng thời theo cơ chế tương tác. Để nâng cao hiệu suất sử dụng CPU, cần cho phép chế độ đa chương mà trong đó các quá trình chia sẻ CPU với nhau để hoạt động đồng hành.

V.3 Hệ thống đa chương với phân khu cố định

Một trong những phương pháp đơn giản nhất để cấp phát bộ nhớ là chia bộ nhớ thành những phân khu có kích thước cố định. Mỗi phân khu có thể chứa chính xác một quá trình. Do đó, cấp độ đa chương được giới hạn bởi số lượng phân khu. Trong phương pháp đa phân khu, khi một phân khu rảnh, một quá trình được chọn từ hàng đợi nhập và được nạp vào phân khu trống. Khi quá trình kết thúc, phân khu trở nên sẵn dùng cho một quá trình khác. Có hai tiếp cận để tổ chức hàng đợi:

- **Sử dụng nhiều hàng đợi**: mỗi phân khu sẽ có một hàng đợi tương ứng (hình VII-7a). Khi một quá trình mới được tạo ra, nó được đưa vào hàng đợi của phân khu có kích thước nhỏ nhất thoả nhu cầu chứa nó. Cách tổ chức này có khuyết điểm trong trường hợp các hàng đợi của một số phân khu trống trong khi các hàng đợi của các phân khu khác lại đầy, buộc các quá trình trong những hàng đợi này phải chờ được cấp phát bộ nhớ.
- **Sử dụng một hàng đợi**: tất cả các quá trình được đặt trong hàng đợi duy nhất (hình VII-7b). Khi có một phân khu trống, quá trình đầu tiên trong hàng đợi có kích thước phù hợp sẽ được đặt vào phân khu và cho xử lý.



a. Sử dụng nhiều hàng đợi b. Sử dụng một hàng đợi

Hình 0-7 Cấp phát phân khu có kích thước cố định

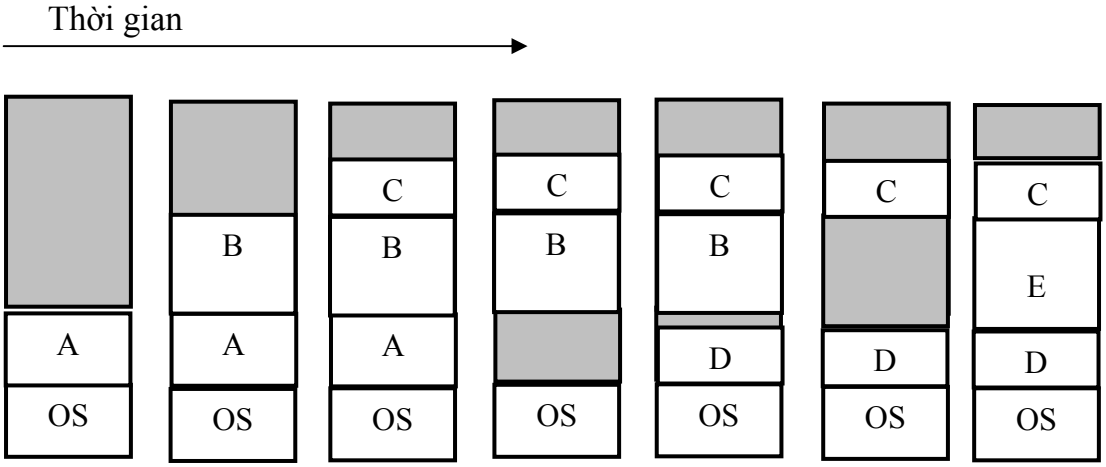
Khi sử dụng giải thuật này người ta muốn tránh sự hao phí một phân khu lớn cho một công việc nhỏ, nhưng lại xảy ra bất bình đẳng, bất lợi đối với các công việc nhỏ. Để giải quyết người ta thêm vào qui luật là một công việc sẽ không bị bỏ qua nữa nếu nó đã bị bỏ qua **k** lần qui định. Mỗi lần một công việc bị bỏ qua nó được đánh dấu một điểm. Khi đạt được số điểm qui định, nó sẽ không bị bỏ qua nữa, sẽ được nạp vào và thực hiện mặc dầu có thể trên một phân khu lớn hơn.

Phương pháp này ban đầu được sử dụng bởi hệ điều hành IBM OS/360, nó được gọi là MFT (Multiprogramming with Fixed number of Tasks). Hiện nay nó không còn sử dụng nữa.

V.4 Hệ thống đa chương với phân khu động

Cơ chế này là tổng quát của cơ chế phân khu cố định. Nó được dùng chủ yếu trong môi trường xử lý theo lô. Nhiều ý tưởng được trình bày ở đây cũng có thể áp dụng tới môi trường chia thời mà trong đó phân đoạn thuần được dùng cho việc quản lý bộ nhớ.

Hệ điều hành giữ một bảng hiển thị những phần nào của bộ nhớ là sẵn dùng và phần nào đang bận. Ban đầu, tất cả bộ nhớ là sẵn dùng cho quá trình người dùng, và được xem như một khối lớn bộ nhớ sẵn dùng hay một lỗ. Khi một quá trình đến và cần bộ nhớ, chúng ta tìm kiếm một lỗ trống đủ lớn cho quá trình này. Nếu chúng ta tìm thấy, chúng ta cấp phát chỉ phần bộ nhớ nhiều bằng lượng được yêu cầu, phần còn lại sẵn dùng để thoả mãn những yêu cầu tương lai (Hình VII-8).



Hình VIII-8 Cấp phát các phân khu có kích thước thay đổi

Khi các quá trình đi vào hệ thống, chúng được đặt vào hàng đợi nhập. Hệ điều hành xem xét yêu cầu bộ nhớ của mỗi quá trình và lượng không gian bộ nhớ sẵn có để xác định các quá trình nào được cấp phát bộ nhớ. Khi một quá trình được cấp không gian, nó được nạp vào bộ nhớ và sau đó nó có thể cạnh tranh CPU. Khi một quá trình kết thúc, nó giải phóng bộ nhớ của nó, sau đó hệ điều hành có thể đặt một quá trình khác từ hàng đợi nhập.

Tại bất cứ thời điểm được cho, chúng ta có một danh sách kích thước khối trống và hàng đợi nhập. Hệ điều hành có thể xếp hàng đợi nhập dựa theo giải thuật định thời. Bộ nhớ được cấp phát tới quá trình cho đến khi các yêu cầu bộ nhớ của quá trình kế tiếp không thể được thoả; không có khối bộ nhớ trống (hay lỗ) đủ lớn để quản lý quá trình đó. Sau đó, hệ điều hành có thể chờ cho đến khi khối đủ lớn sẵn dùng hay nó có thể di chuyển xuống hàng đợi nhập để xem các yêu cầu bộ nhớ nhỏ hơn của các quá trình khác có thể được thoả hay không.

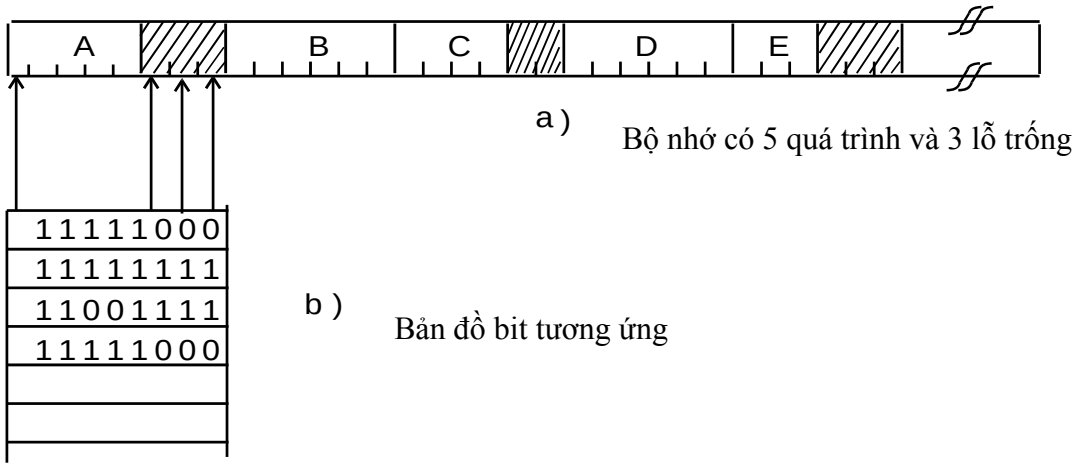
Thông thường, một tập hợp các lỗ có kích thước khác nhau được phân tán khắp bộ nhớ tại bất cứ thời điểm được cho. Khi một quá trình đến và yêu cầu bộ nhớ, hệ thống tìm tập hợp này một lỗ trống đủ lớn cho quá trình này. Nếu lỗ trống quá lớn, nó được chia làm hai: một phần được cấp tới quá trình đến; phần còn lại được trả về tập hợp các lỗ. Nếu lỗ mới nằm kề với các lỗ khác, các lỗ nằm kề này được gom lại để tạo thành một lỗ lớn hơn. Tại thời điểm này, hệ thống cần kiểm tra có quá trình nào đang chờ bộ nhớ và bộ nhớ trống mới hay bộ nhớ vừa được kết hợp lại có thể thoả yêu cầu của bất cứ quá trình đang chờ này không.

Thủ tục này là một trường hợp đặc biệt của **vấn đề cấp phát lưu trữ động** là làm cách nào để thoả mãn một yêu cầu có kích thước n từ danh sách lỗ trống. Có hai giải pháp chủ yếu cho vấn đề này.

1) **Quản lý bằng bản đồ bit**: bộ nhớ được chia thành các đơn vị cấp phát, mỗi đơn vị được ánh xạ tới một bit trong bản đồ bit (Hình VII-9). Giá trị bit này xác định trạng thái của đơn vị bộ nhớ đó: 0 đang tự do, 1 đã được cấp phát. Khi cần nạp một quá trình có kích thước k đơn vị, hệ thống sẽ tìm trong bản đồ bit một dãy k bit có giá trị 0.

Kích thước của đơn vị cấp phát là vấn đề lớn trong thiết kế. Nếu kích thước đơn vị cấp phát nhỏ sẽ làm tăng kích thước của bản đồ bit. Ngược lạ, nếu kích thước đơn

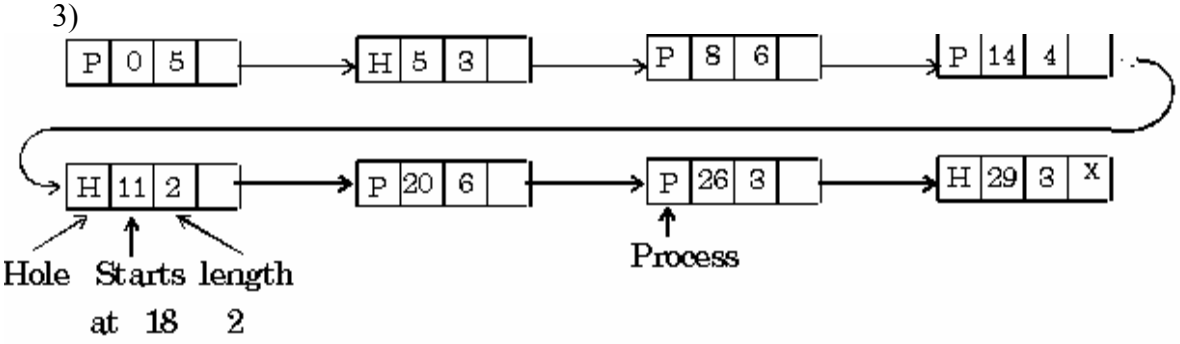
vị cấp phát lớn có thể gây hao phí cho đơn vị cấp phát sau cùng. Đây là giải pháp đơn giản nhưng thực hiện chậm nên ít được dùng.



Hình 0-9 Quản lý bộ nhớ bằng bản đồ bit

2) **Quản lý bằng danh sách liên kết**: dùng một danh sách liên kết để quản lý các phân đoạn bộ nhớ đã cấp phát và phân đoạn tự do, một phân đoạn có thể là một quá trình hay một vùng nhớ trống giữa hai quá trình. Danh sách liên kết gồm nhiều mục từ liên tiếp. Mỗi mục từ gồm 1 bit đầu để xác định phân đoạn đó là lỗ trống (H) hay một quá trình (P), sau đó là 3 từ để chỉ địa chỉ bắt đầu, chiều dài và chỉ điểm tới mục kế tiếp. Việc sắp xếp các phân đoạn theo địa chỉ hay theo kích thước tuỳ thuộc vào giải thuật quản lý bộ nhớ. Sơ đồ quản lý bằng danh sách liên kết tương ứng với sơ đồ quản lý bằng bản đồ bit được minh hoạ trong hình VII-10.

3)



Hình 0-10 Quản lý bộ nhớ bằng danh sách liên kết

Tập hợp các lỗ trống được tìm thấy để xác định lỗ nào là tốt nhất để cấp phát. Các chiến lược first-fit, best-fit, worst-fit là những chiến lược phổ biến nhất được dùng để chọn một lỗ trống từ tập hợp các lỗ trống.

- **First-fit**: cấp phát lỗ trống đầu tiên đủ lớn. Tìm kiếm có thể bắt đầu tại đầu tập hợp các lỗ trống hay tại điểm kết thúc của tìm kiếm first-fit trước đó. Chúng ta dừng tìm kiếm ngay khi chúng ta tìm thấy một lỗ trống đủ lớn.
- **Best-fit**: cấp phát lỗ trống nhỏ nhất đủ lớn. Chúng ta phải tìm toàn bộ danh sách, trừ khi danh sách được xếp thứ tự theo kích thước. Chiến lược này tạo ra lỗ trống nhỏ nhất còn thừa lại.

- **Worst-fit**: cấp phát lỗ trống lớn nhất. Chúng ta phải tìm toàn danh sách trừ khi nó được xếp theo thứ tự kích thước. Chiến lược này tạo ra lỗ trống còn lại lớn nhất mà có thể có ích hơn lỗ trống nhỏ từ tiếp cận best-fit.

Các mô phỏng hiển thị rằng cả first-fit và best-fit là tốt hơn worst-fit về việc giảm thời gian và sử dụng lưu trữ. Giữa first-fit và best-fit không thể xác định rõ chiến lược nào tốt hơn về sử dụng lưu trữ, nhưng first-fit có tốc độ nhanh hơn.

Tuy nhiên, các giải thuật này gặp phải vấn đề **phân mãnh ngoài** (external fragmentation). Khi các quá trình được nạp và được xoá khỏi bộ nhớ, không gian bộ nhớ trống bị phân rã thành những mãnh nhỏ. Phân mãnh ngoài tồn tại khi tổng không gian bộ nhớ đủ để thoả mãn một yêu cầu, nhưng nó không liên tục; vùng lưu trữ bị phân mãnh thành một số lượng lớn các lỗ nhỏ. Vấn đề phân mãnh này có thể rất lớn. Trong trường hợp xấu nhất, chúng có thể có một khối bộ nhớ trống nằm giữa mỗi hai quá trình. Nếu tất cả bộ nhớ này nằm trong một khối trống lớn, chúng ta có thể chạy nhiều quá trình hơn.

Chọn lựa first-fit so với best-fit có thể ảnh hưởng tới lượng phân mãnh. (First-fit là tốt hơn đối với một số hệ thống, ngược lại best fit là tốt hơn cho một số hệ thống khác). Một yếu tố khác là phần cuối của khối trống nào được cấp phát. (phần còn dư nào-phần trên đỉnh, hay phần ở dưới đáy?). Vấn đề không do giải thuật nào được dùng mà do phân mãnh ngoài.

V.5 Quản lý bộ nhớ với hệ thống bạn thân

Như ta đã thấy trong phần trước, việc quản lý các lỗ hổng trên những bảng liệt kê được sắp xếp theo kích thước làm cho việc cấp phát bộ nhớ rất nhanh, nhưng lại làm chậm cho việc ngưng cấp phát bởi vì ta phải chú ý đến các láng giềng. **Hệ thống bạn thân** (Buddy System) là một giải thuật quản lý bộ nhớ tận dụng thuận lợi của việc máy tính dùng những số nhị phân cho việc đánh địa chỉ để tăng tốc độ kết hợp những lỗ hổng sát nhau khi một quá trình hoàn thành hoặc được hoán vị ra ngoài.

Với phương pháp này, bộ quản lý bộ nhớ sẽ có một bảng liệt kê những khối còn tự do có kích thước 1, 2, 4, 16...bytes đến kích thước của bộ nhớ, tức là có kích thước bằng lũy thừa của 2. Khi có một quá trình cần cấp phát bộ nhớ, một lỗ hổng có kích thước bằng luỹ thừa của 2 đủ chứa quá trình sẽ được cấp phát. Nếu không có lỗ hổng yêu cầu, các lỗ hổng sẽ được phân đôi cho đến khi có được lỗ hỗng cần thiết. Khi một quá trình chấm dứt, các lỗ hổng kế nhau có kích thước bằng nhau sẽ được nhập lại để tạo thành lỗ hổng lớn hơn. Do đó, giải thuật này được gọi là hệ thống bạn thân.

Thí du: với bộ nhớ 1M, cần phải có 21 bảng liệt kê như thế sắp từ 1 bytes (2^0) đến 1 byte (2^{20}). Khởi đầu toàn bộ bộ nhớ còn tự do và bảng liệt kê 1M có một mục từ độc nhất chứa đựng một lỗ hổng 1M, tất cả các bảng liệt kê khác đều rỗng. Cấu hình bộ nhớ lúc khởi đầu được chỉ ra trong hình VII-11.

0	128 K	256 K	384 K	512K	640 K	768 K	896 K	1M	
Initially									**1 Hole**
request 70	**A**	**128**		**256**			**512**		**3**
request 35	**A**	**B**	**64**	**256**			**512**		**3**
request 80	**A**	**B**	**64**	**C**	**128**		**512**		**3**
return A	**128**	**B**	**64**	**C**	**128**		**512**		**4**

request 60	**128**	**B**	**D**	**C**	**128**	**512**	**4**
return B	**128**	**64**	**D**	**C**	**128**	**512**	**4**
return D	**256**			**C**	**128**	**512**	**3**
return C	**1024**						**1**

Hình 0-11Hệ thống bạn thân với kích thước 1M

Bây giờ chúng ta hãy xem cách hệ thống buddy làm việc khi một quá trình 70K được hoán vị vào bộ nhớ trống 1M. Do những lỗ hổng chỉ có thể có kích thước là lũy thừa của 2, 128K sẽ được yêu cầu, bởi vì đó chính là lũy thừa nhỏ nhất của 2 đủ lớn. Không có khối 128K sẵn, cũng không có các khối 256K và 512K. Vì vậy khối 1M sẽ được chia làm hai khối 512K, được gọi là những bạn thân; một tại địa chỉ 0 và một tại địa chỉ 512K. Sau đó khối tại địa chỉ thấp hơn, chính là khối tại 0 lại được phân làm hai khối bạn thân 256K, một tại 0 và một tại 256K. Cái thấp hơn của chúng lại được phân làm hai khối 128K, và khối tại 0, đánh dấu là A trong hình được cấp phát cho quá trình.

Kế đến, một quá trình 35K được hoán vị vào. Khi đó ta cần khối 64K, nhưng cũng không có sẵn, vì thế phải phân phối khối 128K thành hai khối bạn thân 64K, một tại địa chỉ 128K, một tại 192K. Khối tại 128K được cấp cho quá trình, trong hình là B. Yêu cầu thứ ba là 80K.

Bây giờ ta hãy xem những gì xảy ra khi một khối được trả lại. Giả sử tại thời điểm này khối 128K (mà chỉ dùng có 70K) được tự do. Khi đó khối 128K sẽ được đưa vào bảng tự do. Bây giờ cần một khối 60K. Sau khi kiểm tra, khối 64K tại 192K được cấp phát và nó được đánh dấu là C.

Bây giờ khối B được trả lại. Tại thời điểm này có hai khối 128K tự do nhưng chúng không được kết hợp lại. Chú ý rằng ngay cả khi khối 128K tại 0 được phân ra làm 2, khối tại 9 được dùng và khối tại 84K còn tự do, sự kết hợp cũng không xãy ra. Khi D được trả lại, sẽ có sự kết hợp lại thành khối 256K tại 0. Cuối cùng, khi C được trả lại, sẽ có kết hợp tạo thành 1 lỗ hổng 1M như ban đầu.

Hệ thống bạn thân có sự thuận lợi so với những giải thuật cùng sắp xếp theo kích thước của khối. Sự thuận lợi này là khi có một khối 2^k bytes tự do, bộ quản lý bộ nhớ chỉ cần tìm trong bảng liệt kê lỗ hổng có kích thước 2^k để xem chúng có khả năng kết hợp được hay không. Với những giải thuật khác mà trong đó cho phép các khối bộ nhớ được phân chia một cách tùy ý, việc tìm kiếm phải diễn ra trên tất cả các bảng liệt kê. Do dó, hệ thống bạn thân làm việc nhanh hơn.

Đáng tiếc, nó lại cực kỳ kém hiệu quả trong việc sử dụng bộ nhớ. Một quá trình 35K phải được cấp phát đến 64K, hao phí đến 29K. Sự hao phí này được gọi là sự **phân mảnh trong** (internal fragmentation), bởi vì phần bộ nhớ hao phí nằm bên trong đoạn được cấp phát. Còn trong các phần trước ta thấy những lỗ hổng ở giữa các đoạn, nhưng không có sự hao phí bên trong các đoạn, do đó kiểu này được coi là sự phân mảnh ngoài.

V.6 Phân mãnh

Phân mãnh bộ nhớ có thể là phân mãnh trong hoặc phân mãnh ngoài. Xét cơ chế cấp phát nhiều phân khu với một lỗ trống có kích thước 18,464 bytes. Giả sử rằng quá trình tiếp theo yêu cầu 18,462 bytes. Nếu chúng ta cấp phát chính xác khối được yêu cầu, chúng ta để lại một lỗ trống có kích thước 2 bytes. Chi phí để giữ vết của lỗ này sẽ lớn hơn kích thước của lỗ trống. Tiếp cận thông thường là chia bộ nhớ vật lý thành những khối có kích thước cố định, và cấp phát bộ nhớ dựa theo đơn vị của kích thước khối. Với tiếp cận này, bộ nhớ được cấp phát tới một quá trình có thể là lớn hơn một chút so với khối được yêu cầu. Sự chênh lệnh giữa hai số này là phân mãnh trong-bộ nhớ bị phân mãnh trong đối với một phân khu thì không thể được dùng.

Một giải pháp đối với phân mãnh ngoài là **kết lại thành khối** (compaction). Mục tiêu là di chuyển nội dung bộ nhớ để đặt tất cả bộ nhớ trống với nhau trong một khối lớn. Kết khối không phải luôn thực hiện được. Nếu việc tái định vị là tĩnh và được thực hiện tại thời điểm hợp dịch và nạp thì việc kết khối là không thể thực hiện được. Kết khối chỉ có thể thực hiện được chỉ nếu tái định vị là động và được thực hiện tại thời điểm thực thi. Nếu địa chỉ được tái định vị động, tái định vị yêu cầu chỉ di chuyển chương trình và dữ liệu, sau đó thay đổi thanh ghi nền để phản ánh địa chỉ nền mới. Khi kết khối là có thể, chúng ta phải xác định chi phí của nó. Giải thuật kết khối đơn giản nhất là di chuyển tất cả quá trình tới cuối bộ nhớ; tất cả lỗ trống di chuyển theo hướng ngược lại, tạo ra một lỗ trống lớn của bộ nhớ sẵn dùng. Cơ chế này có thể đắt.

Một giải pháp khác cho vấn đề phân mãnh ngoài là cho phép không gian địa chỉ luận lý của một quá trình là không liên tục, do đó cho phép một quá trình được cấp phát bộ nhớ vật lý bất cứ đâu sau khi sẵn dùng. Hai kỹ thuật bù trừ để đạt giải pháp này là phân trang và phân đoạn.

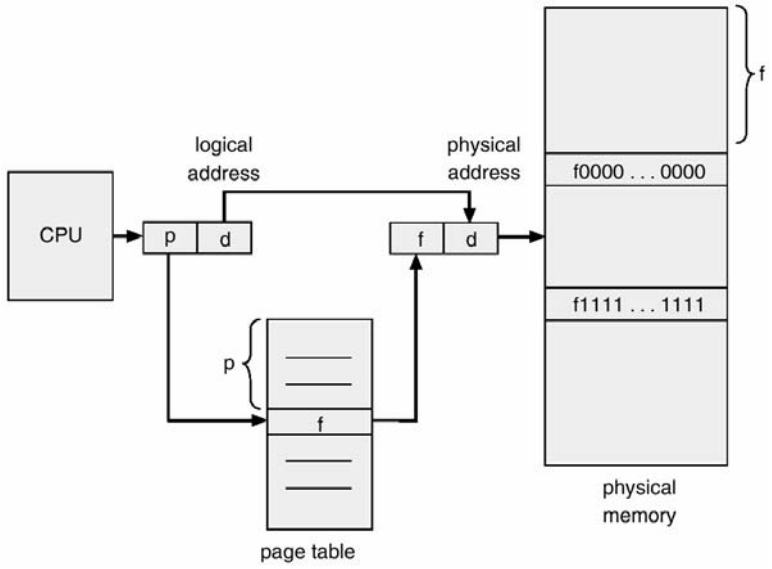
VI Cấp phát không liên tục

VI.1 Phân trang

Phân trang là cơ chế quản lý bộ nhớ cho phép không gian địa chỉ vật lý của quá trình là không kề nhau. Phân trang tránh vấn đề đặt vừa khít nhóm bộ nhớ có kích thước thay đổi vào vùng lưu trữ phụ (backing store) mà hầu hết các cơ chế quản lý bộ nhớ trước đó gặp phải. Khi phân đoạn mã và dữ liệu nằm trong bộ nhớ được hoán vị ra, không gian phải được tìm thấy trên vùng lưu trữ phụ. Vấn đề phân mãnh được thảo luận trong sự kết nối với bộ nhớ chính cũng thông dụng như với vùng lưu trữ phụ, ngoại trừ truy xuất thấp hơn nhiều, vì thế kết khối là không thể. Vì lợi điểm của nó so với các phương pháp trước đó nên phân trang trong những dạng khác nhau được dùng phổ biến trong hầu hết các hệ điều hành.

Về truyền thống, hỗ trợ phân trang được quản lý bởi phần cứng. Tuy nhiên, những thiết kế gần đây cài đặt phân trang bằng cách tích hợp chặt chẻ phần cứng và hệ điều hành, đặc biệt trên các bộ vi xử lý 64-bit.

VI.1.1 Phương pháp cơ bản



Hình 0-12 Phần cứng phân trang

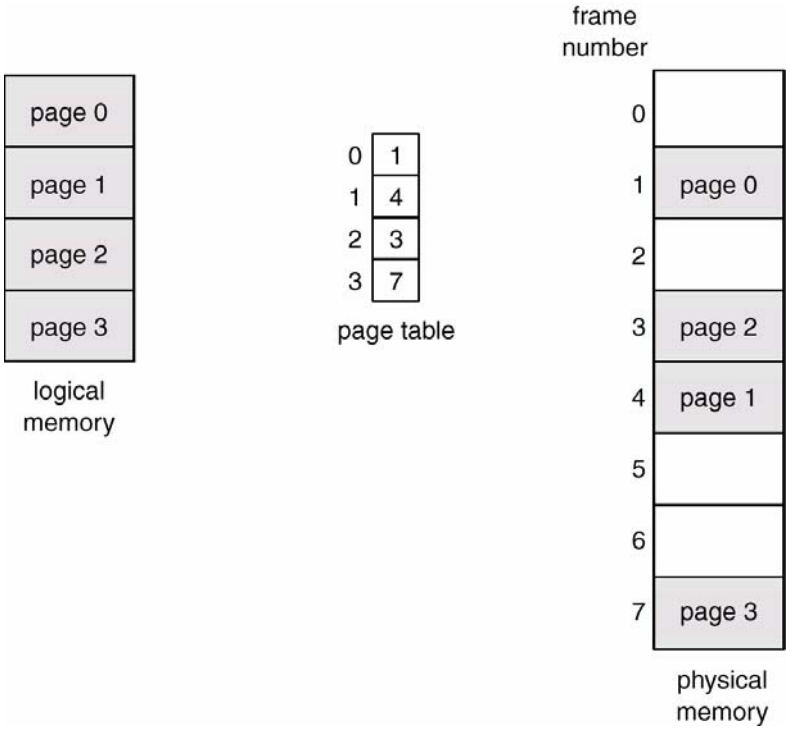
Bộ nhớ vật lý được chia thành các khối có kích thước cố định được gọi là các **khung** (frames). Bộ nhớ luận lý cũng được chia thành các khối có cùng kích thước được gọi là các **trang** (pages). Khi một quá trình được thực thi, các trang của nó được nạp vào các khung bộ nhớ sẵn dùng từ vùng lưu trữ phụ. Vùng lưu trữ phụ được chia thành các khối có kích thước cố định và có cùng kích thước như các khung bộ nhớ.

Hỗ trợ phần cứng cho phân trang được hiển thị trong hình VII-12. Mỗi địa chỉ được tạo ra bởi CPU được chia thành hai phần: số trang (p) và độ dời trang (d). Số trang được dùng như chỉ mục vào bảng trang. Bảng trang chứa địa chỉ nền của mỗi trang trong bộ nhớ vật lý. Địa chỉ nền này được kết hợp với độ dời trang để định nghĩa địa chỉ bộ nhớ vật lý mà nó được gởi đến đơn vị bộ nhớ. Mô hình phân trang bộ nhớ được hiển thị như hình VII-13.

Kích thước trang (giống như kích thước khung) được định nghĩa bởi phần cứng. Kích thước của một trang điển hình là luỹ thừa của 2, từ 512 bytes đến 16MB trên trang, phụ thuộc vào kiến trúc máy tính. Chọn luỹ thừa 2 cho kích thước trang thực hiện việc dịch địa chỉ luận lý thành số trang và độ dời trang rất dễ dàng. Nếu kích thước không gian địa chỉ là 2^m, và kích thước trang là 2^n đơn vị địa chỉ (byte hay từ) thì m-n bits cao của địa chỉ luận lý chỉ số trang, n bits thấp chỉ độ dời trang. Do đó, địa chỉ luận lý như sau:

Số trang	Độ dời trang
P	D
m –n	N

ở đây p là chỉ mục trong bảng trang và d là độ dời trong trang.



Hình 0-13 Mô hình phân trang của bộ nhớ luận lý và vật lý

Thí dụ: xét bộ nhớ trong hình VII-14. Sử dụng kích thước trang 4 bytes và bộ nhớ vật lý 32 bytes (có 8 trang), chúng ta hiển thị cách nhìn bộ nhớ của người dùng có thể được ánh xạ tới bộ nhớ vật lý như thế nào. Địa chỉ luận lý 0 là trang 0, độ dời 0. Chỉ mục trong bảng trang, chúng ta thấy rằng trang 0 ở trong khung 5. Do đó, địa chỉ luận lý 0 ánh xạ tới địa chỉ vật lý 20 (=(5x4)+0). Địa chỉ luận lý 3 (trang 0, độ dời 3) ánh xạ tới địa chỉ vật lý 23 (=(5x4)+3). Địa chỉ luận lý 4 ở trang 1, độ dời 0; dựa theo bảng trang, trang 1 được ánh xạ tới khung 6. Do đó, địa chỉ luận lý 4 ánh xạ tới địa chỉ 24(=(6x4)+0). Địa chỉ luận lý 13 ánh xạ tới địa chỉ vật lý 9.

Có thể chú ý rằng phân trang là một dạng của tái định vị động. Mỗi địa chỉ luận lý được giới hạn bởi phần cứng phân trang tới địa chỉ vật lý. Sử dụng phân trang tương tự sử dụng một bảng các thanh ghi nền (hay tái định vị), một thanh ghi cho mỗi khung bộ nhớ.

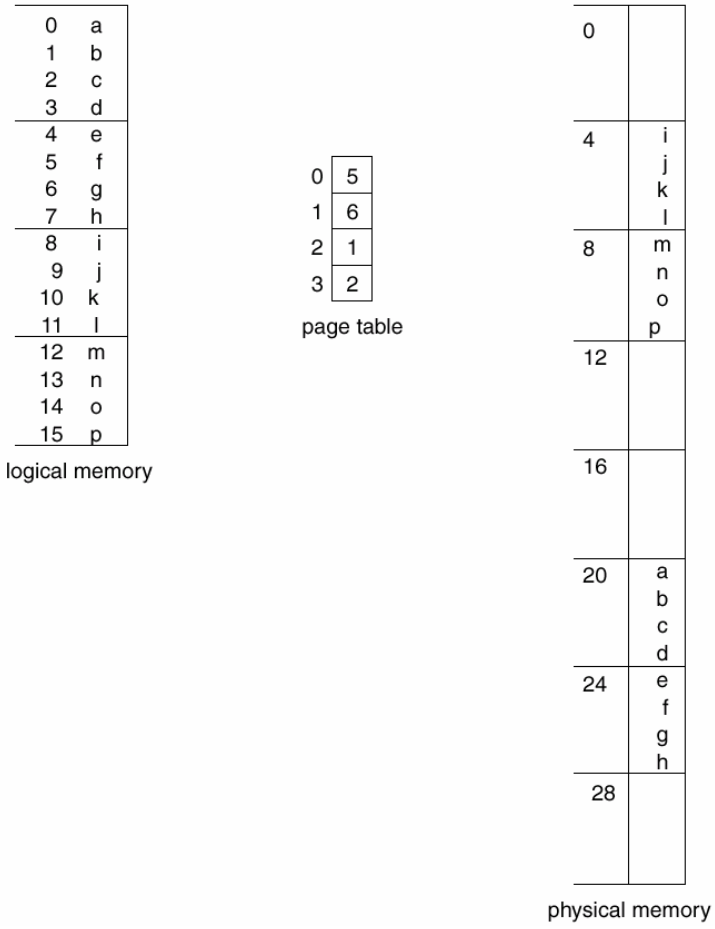
Khi chúng ta sử dụng một cơ chế phân trang, chúng ta không có phân mãnh bên ngoài: bất kỳ khung trống có thể được cấp phát tới một quá trình cần nó. Tuy nhiên, chúng ta có thể có phân mãnh trong. Chú ý rằng các khung được cấp phát như các đơn vị. Nếu các yêu cầu bộ nhớ của một quá trình không xảy ra để rơi trên giới hạn của trang, thì khung cuối cùng được cấp phát có thể không đầy hoàn toàn. Thí dụ, nếu các trang là 2048 bytes, một quá trình 72,766 bytes sẽ cần 35 trang cộng với 1086 bytes. Nó được cấp phát 36 khung, do đó phân mãnh trong là 2048 - 1086 = 962 bytes. Trong trường hợp xấu nhất, một quá trình cần n trang cộng với 1 byte. Nó sẽ được cấp phát n+1 khung, dẫn đến phân mãnh trong gần như toàn bộ khung.

Nếu kích thước quá trình độc lập với kích thước của trang, thì chúng ta mong muốn phân mãnh trong trung bình là ½ trang trên một quá trình. Xem xét này đề nghị rằng kích thước trang nhỏ là mong muốn. Tuy nhiên, chi phí liên quan tới mỗi mục từ bảng trang và chi phí này giảm khi kích thước trang tăng. Vì thế nhập/xuất đĩa là hiệu quả hơn khi số lượng dữ liệu được truyền lớn hơn. Thường thì kích thước trang lớn lên theo thời gian khi các quá trình, tập hợp dữ liệu, bộ nhớ chính trở nên lớn hơn. Ngày nay, các trang điển hình nằm trong khoảng 4 KB tới 8 KB, và một số hệ thống hỗ trợ kích thước trang lớn hơn. CPU và nhân thậm chí hỗ trợ nhiều kích thước khác

nhau. Thí dụ, Solaris dùng 8 KB và 4 MB kích thước trang, phụ thuộc dữ liệu được lưu bởi trang. Hiện nay, các nhà nghiên cứu đang phát triển việc hỗ trợ kích thước trang khác nhau.

Mỗi mục từ bảng trang thường dài 4 bytes, nhưng kích thước có thể thay đổi. Một mục từ 32-bit có thể chỉ tới một khung trang vật lý 2^{32}. Nếu một khung là 4 KB, thì hệ thống với những mục từ 4 bytes có thể đánh địa chỉ cho 2^{44} bytes (hay 16 TB) bộ nhớ vật lý.

Khi một quá trình đi vào hệ thống để được thực thi, kích thước của nó, được diễn tả trong các trang, được xem xét. Mỗi trang của quá trình cần trên một khung. Do đó, nếu quá trình yêu cầu n trang, ít nhất n khung phải sẵn dùng trong bộ nhớ. Nếu n khung là sẵn dùng, chúng được cấp phát tới quá trình đang đi vào này. Trang đầu tiên của quá trình được nạp vào một trong những khung được cấp phát, và số khung được đặt vào trong bảng trang cho quá trình này. Trang kế tiếp được nạp vào một khung khác, và số khung của nó được đặt vào trong bảng trang, …(hình VII-15).

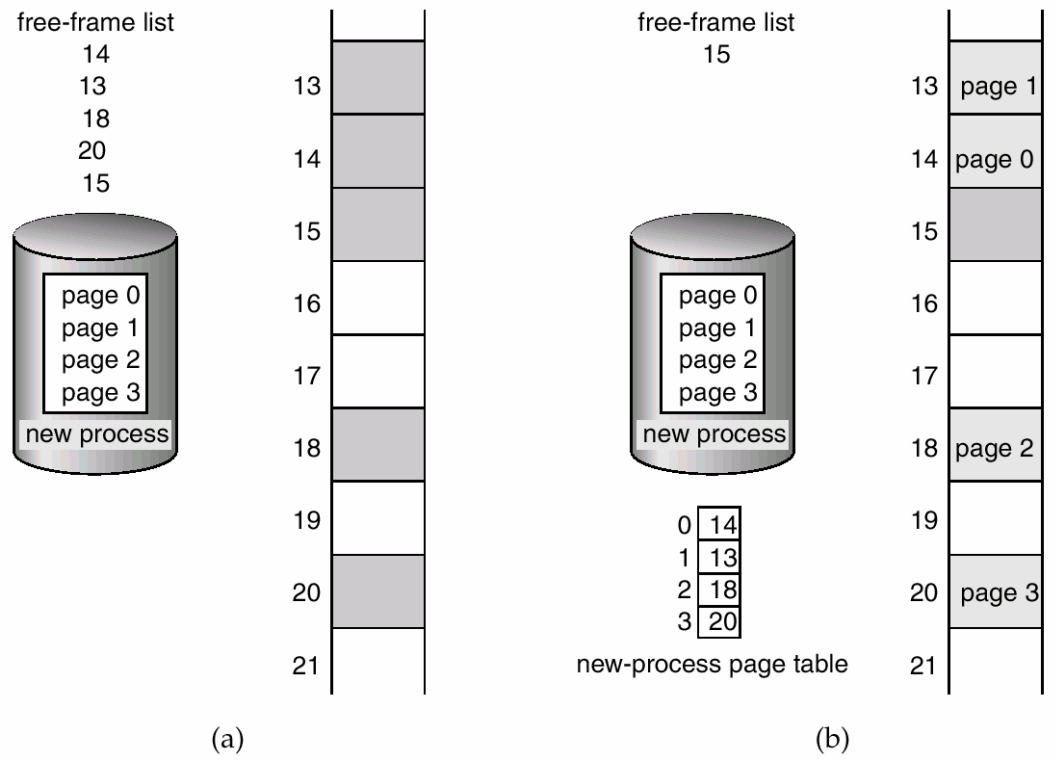


Hình 0-14 Thí dụ phân trang cho bộ nhớ 32 bytes với các trang có kích thức 4 bytes

Một khía cạnh quan trọng của phân trang là sự phân chia rõ ràng giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự. Chương trình người dùng nhìn bộ nhớ như một không gian liên tục, chứa chỉ một chương trình. Sự thật, chương trình người dùng được phân bố khắp bộ nhớ vật lý mà nó cũng quản lý các quá trình khác. Sự khác nhau giữa tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự được làm cho tương thích bởi phần cứng dịch địa chỉ. Địa chỉ luận lý được dịch thành địa chỉ vật lý. Ánh xạ này được che giấu từ người dùng và được điều khiển bởi hệ điều hành. Chú ý rằng như định nghĩa, quá trình người dùng không thể truy xuất bộ nhớ mà nó

không sở hữu. Không có cách định địa chỉ bộ nhớ bên ngoài bảng trang của nó và bảng chứa chỉ những trang mà quá trình sở hữu.

Vì hệ điều hành đang quản lý bộ nhớ vật lý nên nó phải hiểu những chi tiết cấp phát bộ nhớ vật lý; khung nào được cấp phát, khung nào còn trống, tổng khung hiện có là bao nhiêu,…Thông tin này được giữ trong một cấu trúc dữ liệu được gọi là bảng khung. Bảng khung chỉ có một mục từ cho mỗi khung trang vật lý, hiển thị khung trang đó đang rảnh hay được cấp phát. Nếu khung trang được cấp phát, thì xác định trang nào của quá trình nào được cấp.



Hình 0-15 các khung trống. (a) trước khi cấp phát. (b) sau khi cấp phát

Ngoài ra, hệ điều hành phải biết rằng quá trình người dùng hoạt động trong không gian người dùng, và tất cả địa chỉ luận lý phải được ánh xạ để phát sinh địa chỉ vật lý. Nếu người dùng thực hiện lời gọi hệ thống (thí dụ: để thực hiện nhập/xuất) và cung cấp địa chỉ như một tham số (thí dụ: vùng đệm), địa chỉ đó phải được ánh xạ để sinh ra địa chỉ vật lý đúng. Hệ điều hành duy trì một bản sao của bảng trang cho mỗi quá trình, như nó duy trì bản sao của bộ đếm chỉ thị lệnh và nội dung thanh ghi. Bản sao này được dùng để dịch địa chỉ luận lý thành địa chỉ vật lý bất cứ khi nào hệ điều hành phải ánh xạ địa chỉ luận lý tới địa chỉ vật lý dạng thủ công. Nó cũng được dùng bởi bộ phân phát CPU để địa chỉ bảng trang phần cứng khi một quá trình được cấp phát CPU. Do đó, trang gia tăng thời gian chuyển đổi ngữ cảnh.

VI.1.2 Hỗ trợ phần cứng

Mỗi hệ điều hành có phương pháp riêng để lưu trữ các bảng trang. Hầu hết đều cấp phát một bảng trang cho mỗi quá trình. Một con trỏ chỉ tới một bảng trang được lưu trữ với những giá trị thanh ghi thanh ghi khác nhau (giống như bộ đếm chỉ thị lệnh) trong khối điều khiển quá trình. Khi bộ phân phát được yêu cầu bắt đầu một quá trình, nó phải nạp lại các thanh ghi người dùng và định nghĩa các giá trị bảng trang phần cứng phù hợp từ bảng trang người dùng được lưu trữ.

Cài đặt phần cứng của bảng trang có thể được thực hiện trong nhiều cách. Cách đơn giản nhất, bảng trang được cài đặt như tập hợp các thanh ghi tận hiến. Các thanh ghi này nên được xây dựng với tính logic tốc độ rất cao để thực hiện việc dịch địa chỉ trang hiệu quả. Mọi truy xuất tới bộ nhớ phải kiểm tra kỹ lưỡng bảng đồ trang, vì vậy tính hiệu quả là vấn đề xem xét chủ yếu. Bộ phân phát CPU nạp lại các thanh ghi này chỉ khi nó nạp lại các thanh ghi khác. Dĩ nhiên, các chỉ thị để nạp hay hiệu chỉnh các thanh ghi bảng trang phải được cấp quyền để mà chỉ hệ điều hành có thể thay đổi bản đồ bộ nhớ. DEC PDP-11 là một thí dụ về kiến trúc như thế. Địa chỉ chứa 16 bits, và kích thước trang là 8 KB. Do đó, bảng trang chứa 8 mục từ mà chúng được giữ trong các thanh ghi nhanh.

Sử dụng các thanh ghi cho bảng trang chỉ phù hợp nếu bảng trang có kích thước nhỏ (thí dụ: 256 mục từ). Tuy nhiên, hầu hết các máy tính tương thời cho phép bảng trang rất lớn (thí dụ, 1 triệu mục từ). Đối với những máy này, việc sử dụng các thanh ghi nhanh để cài đặt bảng trang là không khả thi. Hay đúng hơn là, bảng trang được giữ trong bộ nhớ chính, và thanh ghi nền bảng trang (page-table base register-PTBR) chỉ tới thanh ghi bảng trang. Thay đổi các bảng trang yêu cầu thay đổi chỉ một thanh ghi, về căn bản cắt giảm thời gian chuyển ngữ cảnh.

Vấn đề với tiếp cận này là thời gian được yêu cầu để truy xuất vị trí bộ nhớ người dùng. Nếu chúng ta muốn truy xuất vị trí i, đầu tiên chúng ta phải xác định chỉ mục trong bảng trang, sử dụng giá trị trong độ dời PTBR bởi số trang cho i. Tác vụ này yêu cầu một truy xuất bộ nhớ. Nó cung cấp chúng ta số khung được nối kết với độ dời trang để sinh ra địa chỉ thực. Sau đó, chúng ta có thể truy xuất tới nơi được mong muốn trong bộ nhớ. Với cơ chế này, hai truy xuất bộ nhớ được yêu cầu để truy xuất một byte (một cho mục từ bảng trang, một cho byte đó). Do đó, truy xuất bộ nhớ bị chậm bởi một trong hai yếu tố đó. Sự trì hoãn này không thể chấp nhận dưới hầu hết trường hợp vì thế chúng ta phải sử dụng đến hoán vị!

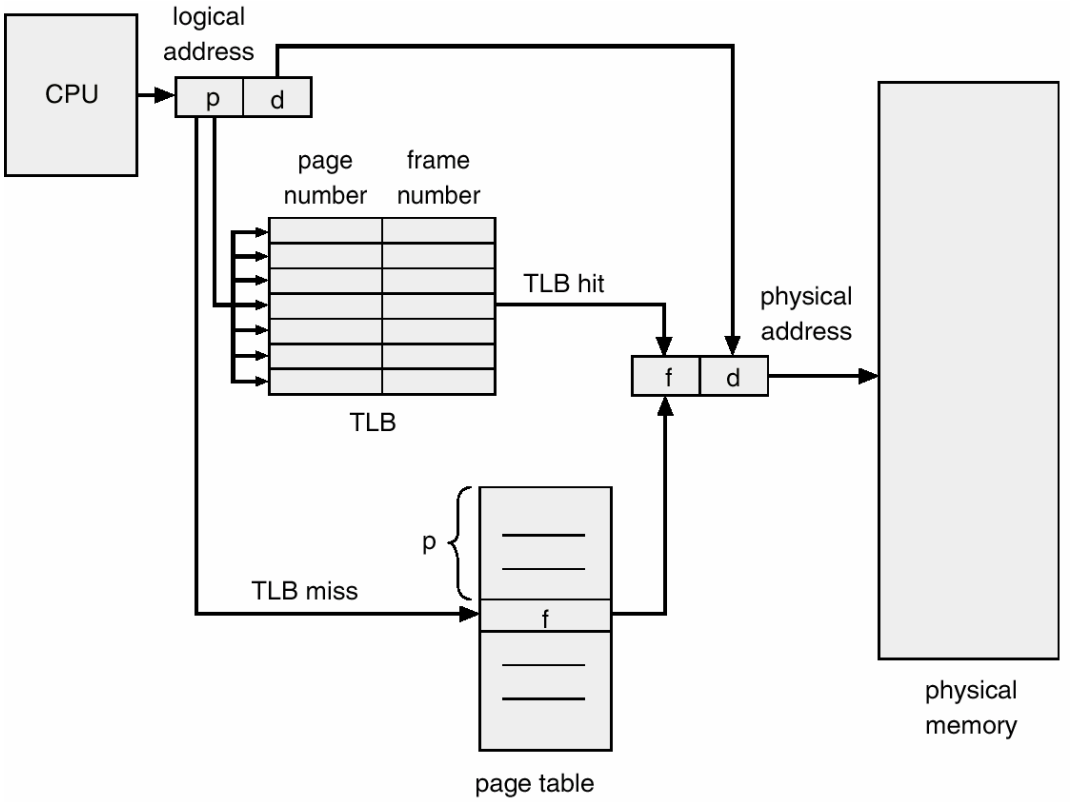
Giải pháp chuẩn cho vấn đề này là dùng bộ lưu trữ (cache) phần cứng đặc biệt, nhỏ, tìm kiếm nhanh được gọi là translation look-aside buffer (TLB). TLB là bộ nhớ kết hợp tốc độ cao. Mỗi mục từ trong TLB chứa hai phần: một khoá (key) và một giá trị (value). Khi bộ nhớ kết hợp được biểu diễn với một thành phần, nó được so sánh với tất cả khoá cùng một lúc. Nếu thành phần được tìm thấy, trường giá trị tương ứng được trả về. Tìm kiếm nhanh nhưng phần cứng đắt. Điển hình, số lượng mục từ trong TLB nhỏ, thường từ 64 đến 1024.

TLB được dùng với các bảng trang trong cách sau. TLB chứa chỉ một vài mục từ bảng trang. Khi một địa chỉ luận lý được phát sinh bởi CPU, số trang của nó được hiện diện trong TLB. Nếu số trang được tìm thấy, khung của nó lập tức sẵn dùng và được dùng để truy xuất bộ nhớ. Toàn bộ tác vụ có thể mất ít hơn 10% thời gian nếu dùng tham chiếu bộ nhớ không được ánh xạ.

Nếu số trang không ở trong TLB (còn gọi là mất TLB), tham chiếu bộ nhớ tới bảng trang phải được thực hiện. Khi số khung đạt được, chúng ta có thể dùng nó để truy xuất bộ nhớ (như hình VII-16). Ngoài ra, chúng ta thêm số trang và số khung tới TLB để mà chúng có thể được tìm thấy nhanh trên tham chiếu tiếp theo. Nếu một TLB đã đầy các mục từ, hệ điều hành phải chọn một mục từ để thay thế. Các chính sách thay thế rất đa dạng từ ít được sử dụng gần đây nhất (least recently used-LRU) tới chọn ngẫu nhiên. Ngoài ra, một số TLB cho phép các mục từ được ***wired down***. Nghĩa là, chúng không thể được xoá khỏi TLB. Điển hình, các mục từ cho nhân thường được wired down.

Một số TLB lưu trữ các định danh không gian địa chỉ (address-space identifers-ASID) trong mỗi mục từ của TLB. Một ASID định danh duy nhất mỗi quá trình và được dùng để cung cấp việc bảo vệ không gian địa chỉ cho quá trình đó. Khi TLB cố

gắng phân giải các số trang ảo, nó đảm bảo ASID cho mỗi quá trình hiện đang chạy trùng khớp với ASID được nối kết với trang ảo. Nếu các ASID không khớp, chúng được xem như mất TLB. Ngoài ra, để cung cấp việc bảo vệ không gian địa chỉ, một ASID cho phép TLB chứa các mục từ cho nhiều quá trình khác nhau cùng một lúc. Nếu TLB không hỗ trợ các ASID riêng thì mỗi lần một bảng trang được chọn (thí dụ, mỗi chuyển ngữ cảnh), một TLB phải được đẩy (hay được xoá) để đảm bảo rằng các quá trình đang thực thi tiếp theo không sử dụng thông tin dịch sai. Ngược lại, có những mục từ cũ trong TLB chứa các địa chỉ ảo nhưng có các địa chỉ không đúng hay không hợp lệ để lại từ quá trình trước.



Hình 0-16 phần cứng phân trang với TBL

Phần trăm thời gian mà số trang xác định được tìm thấy trong TLB được gọi là tỉ lệ chập (hit ratio). Tỉ lệ chập 80% có nghĩa là chúng ta tìm số trang mong muốn trong TLB 80% thời gian. Nếu mất 20 nano giây để tìm TLB và 100 nano giây để truy xuất bộ nhớ, thì một truy xuất bộ nhớ được ánh xạ mất 120 nano giây khi số trang ở trong TLB. Nếu chúng ta không tìm số trang trong TLB (20 nano giây) thì trước hết chúng ta phải truy xuất bộ nhớ cho bảng trang và số khung (100 nano giây), thì sau đó truy xuất byte mong muốn trong bộ nhớ (100 nano giây), tổng thời gian là 220 nano giây. Để tìm thời gian truy xuất bộ nhớ hiệu quả, chúng ta phải đo mỗi trường hợp với xác suất của nó:

Thời gian truy xuất hiệu quả = 0.80 x 120 + 0.2 x 220 = 140 nano giây

Trong thí dụ này, chúng ta gặp phải 40% chậm lại trong thời gian truy xuất bộ nhớ (từ 100 tới 140 nano giây).

Đối với một tỉ lệ chậm 98%, chúng ta có:

Thời gian truy xuất hiệu quả = 0.98 x 120 + 0.02 x 220 = 122 nano giây

Tỉ lệ chập được tăng này chỉ tạo ra 22% chậm lại trong thời gian truy xuất.

VI.1.3 Sự bảo vệ

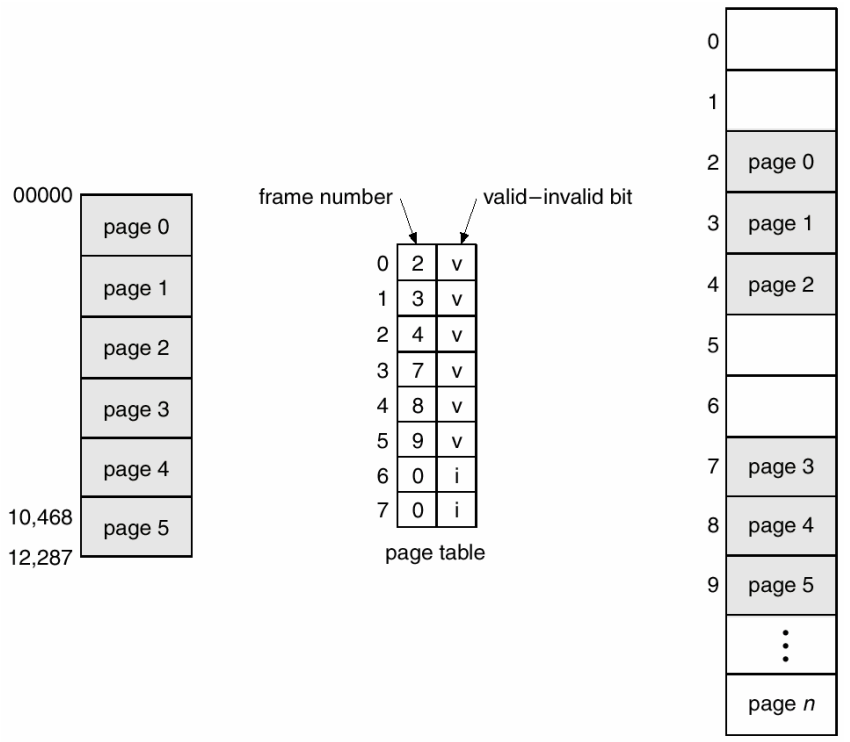
Bảo vệ bộ nhớ trong môi trường phân trang được thực hiện bởi các bit bảo vệ gán liền với mỗi khung. Thông thường, các bit này được giữ trong bảng trang. Một bit có thể định nghĩa một trang để đọc-viết hay chỉ đọc. Mỗi tham chiếu tới bộ nhớ sẽ tìm khắp bảng trang để xác định số khung tương ứng. Tại cùng thời điểm địa chỉ vật lý được tính , các bit bảo vệ có thể được kiểm tra để thẩm định rằng không có thao tác viết nào đang được thực hiện tới trang chỉ đọc. Cố gắng viết tới một trang chỉ đọc gây ra một trap phần cứng tới hệ điều hành (hay xung đột bộ nhớ bảo vệ).

Chúng ta có thể dễ dàng mở rộng tiếp cận này để cung cấp một cấp độ bảo vệ chi tiết hơn. Chúng ta có thể tạo phần cứng để cung cấp bảo vệ chỉ đọc, đọc viết, chỉ thực thi. Hay bằng cách cung cấp các bit bảo vệ riêng cho mỗi loại truy xuất, chúng ta có thể cho phép bất cứ kết hợp của các truy xuất này; các cố gắng không hợp lệ sẽ được trap tới hệ điều hành.

Một bit nữa thường được gán tới mỗi mục từ trong bảng trang: một **bit hợp lệ-không hợp lệ**. Khi bit này được đặt là "hợp lệ" thì giá trị này hiển thị rằng trang được gán trong không gian địa chỉ luận lý bộ nhớ là trang hợp lệ. Nếu bit này được đặt là "không hợp lệ", giá trị này hiển thị trang đó không ở trong không gian địa chỉ luận lý của quá trình. Các địa chỉ không hợp lệ được trap bằng cách sử dụng bit hợp lệ-không hợp lệ. Hệ điều hành thiết lập bit này cho mỗi trang để cho phép hay không cho phép truy xuất tới trang này. Thí dụ, trong một hệ thống với không gian địa chỉ 14 bit (0 tới 16383), chúng ta có thể có một chương trình sử dụng chỉ địa chỉ 0 tới 10468. Cho kích thước trang 2KB, chúng ta xem trường hợp trong hình VII-17. Địa chỉ trong các trang 0, 1, 2, 3, 4, và 5 thường được ánh xạ khắp bảng trang. Tuy nhiên, bất cứ những nỗ lực để tạo ra một địa chỉ trong trang 6 hay 7 nhận thấy rằng bit hợp lệ-không hợp lệ được đặt là không hợp lệ và máy tính sẽ trap tới hệ điều hành (tham chiếu trang không hợp lệ).

Vì chương trình mở rộng chỉ tới địa chỉ 10468, bất cứ tham chiếu vượt ra ngoài địa chỉ đó là không hợp lệ. Tuy nhiên, các tham chiếu tới trang 5 được xem là hợp lệ vì thế những địa chỉ tới 12287 là hợp lệ. Chỉ những địa chỉ từ 12288 tới 16383 là không hợp lệ. Vấn đề này là do kích thước trang 2KB và phản ánh phân mãnh trong của việc phân trang.

Rất hiếm khi một quá trình dùng tất cả dãy địa chỉ của nó. Thật vậy, nhiều quá trình dùng chỉ một phần nhỏ không gian địa chỉ còn trống cho chúng. Nó sẽ lãng phí rất nhiều trong những trường hợp này để tạo một bảng trang với các mục từ cho mỗi trang trong dãy địa chỉ. Hầu hết bảng này sẽ không được dùng nhưng sẽ mang đến không gian bộ nhớ có giá trị. Một số hệ thống cung cấp phần cứng, trong dạng một thanh ghi có chiều dài bảng trang (page-table length register-PTLR) để hiển thị kích thước của bảng trang. Giá trị này được kiểm tra dựa trên mỗi địa chỉ luận lý để thẩm định địa chỉ đó nằm trong dãy địa chỉ hợp lệ cho quá trình. Lỗi của việc kiểm tra này gây ra một trap lỗi tới hệ điều hành.



Hình 0-17 Bit hợp lệ (v) và không hợp lệ (i) trong một bảng trang

VI.1.4 Cấu trúc bảng trang

Trong phần này chúng ta sẽ xem xét một số kỹ thuật thông dụng nhất để xây dựng cấu trúc bảng trang.

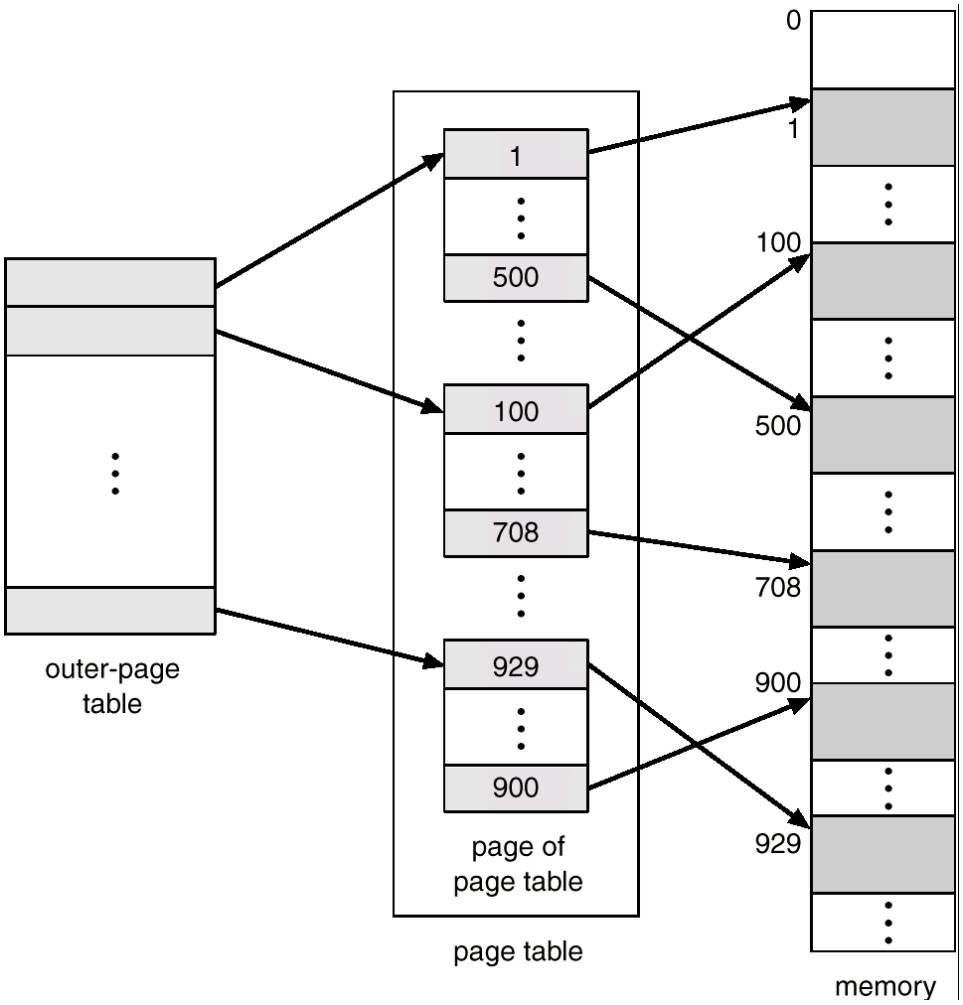
.VI.1.4.1 Bảng trang phân cấp

Hầu hết các hệ thống máy tính hiện đại hỗ trợ một không gian địa chỉ luận lý lớn (2^{32} tới 2^{64}). Trong môi trường như thế, bảng trang trở nên quá lớn. Thí dụ, xét một hệ thống với không gian địa chỉ luận lý 32 bit. Nếu kích thước trang 4KB thì bảng trang có thể chứa tới 1 triệu mục từ ($2^{32}/2^{12}$). Giả sử rằng mỗi mục từ chứa 4 bytes, mỗi quá trình có thể cần tới 4MB không gian địa chỉ vật lý cho một bảng trang. Rõ ràng, chúng ta sẽ không muốn cấp phát bảng trang liên tiếp nhau. Một giải pháp đơn giản cho vấn đề này là chia bảng trang thành những phần nhỏ hơn. Có nhiều cách để đạt được sự phân chia này.

Một cách là dùng giải thuật phân trang hai cấp, trong đó bảng trang cũng được phân trang như hình VII-18.

Đây là thí dụ cho máy 32 bit với kích thước trang 4KB. Địa chỉ luận lý được chia thành số trang chứa 20 bit và độ dời trang chứa 12 bit. Vì chúng ta phân trang bảng trang, số trang được chia thành số trang 10 bit và độ dời trang 10-bit. Do đó, một địa chỉ luận lý như sau:

Số trang		Độ dời trang
P_1	P_2	d
10	10	12



Hình 0-18 Cơ chế bảng trang hai cấp

Ở đây p_1 là chỉ mục trong bảng trang bên ngoài và p_2 là độ dời trong trang của bảng trang bên ngoài. Phương pháp dịch địa chỉ cho kiến trúc này được hiển thị trong hình VII-19. Vì dịch địa chỉ thực hiện từ những phần trong bảng trang bên ngoài, cơ chế này cũng được gọi là **bảng trang được ánh xạ chuyển tiếp** (forward-mapped page table). Petium-II sử dụng kiến trúc này.

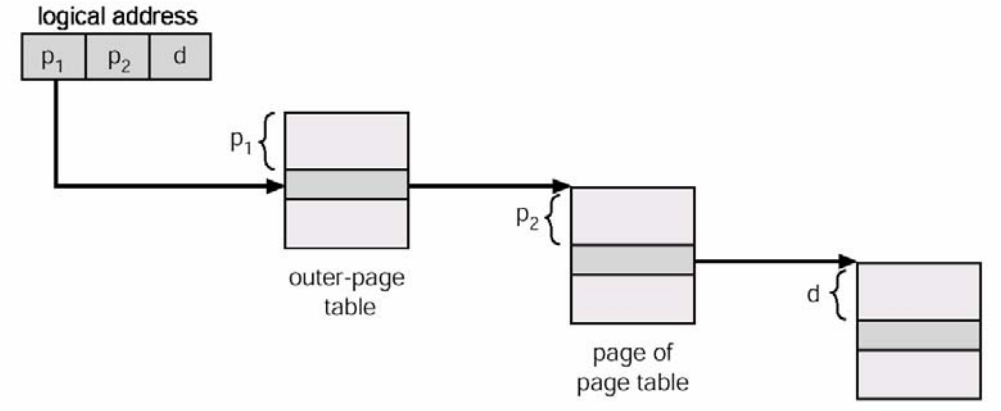
Kiến trúc VAX cũng hỗ trợ một biến dạng của phân trang hai cấp. VAX là máy 32-bit với kích thước trang 512 bytes. Không gian địa chỉ luận lý của một quá trình được chia làm 4 phần bằng nhau, mỗi phần chứa 2^{30} bytes. Mỗi phần biểu diễn một phần khác nhau của không gian địa chỉ luận lý của một quá trình. Hai bit cao đầu tiên của địa chỉ luận lý chỉ rõ phần tương ứng. 21 bits tiếp theo biểu diễn số trang luận lý của phần đó, và 9 bits cuối biểu diễn độ dời trong trang mong muốn. Bằng cách chia bảng trang như thế, hệ điều hành có thể để những phân khu không được dùng cho tới khi một quá trình yêu cầu chúng. Một địa chỉ trên kiến trúc VAX như sau:

Phần	Trang	Độ dời
S	P	D
2	21	9

ở đây s chỉ rõ số phần, p là chỉ mục trong bảng trang và d là độ dời trong trang.

Kích thước của bảng trang cấp một cho một quá trình VAX dùng một phần vẫn là 2^{21} bits * 4 bytes/trang = 8 MB. Để việc sử dụng bộ nhớ chính bị giảm nhiều hơn, VAX phân trang các bảng trang quá trình người dùng.

Đối với các hệ thống có không gian địa chỉ luận lý 64 bits, cơ chế phân trang hai cấp không còn phù hợp nữa. Để thể hiện điểm này, chúng ta giả sử rằng kích thước trang trong hệ thống là 4 KB (2^{12}). Trong trường hợp này, bảng trang sẽ chứa tới 2^{52} mục từ. Nếu chúng ta dùng cơ chế phân trang hai cấp thì các bảng bên trong có thể là một trang dài chứa 2^{10} mục từ. Các địa chỉ sẽ như thế này:



Hình 0-19 Dịch địa chỉ cho kiến trúc phân trang hai cấp 32-bit

Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	D
42	10	12

Bảng trang bên ngoài sẽ chứa 242 mục từ, hay 244 bytes. Các phương pháp được chú trọng để tránh để trang lớn là chia bảng trang bên ngoài thành những phần nhỏ hơn. Tiếp cận này cũng được dùng trên một vài bộ xử lý 32-bit để thêm khả năng mềm dẽo và hiệu quả.

Chúng ta có thể chia bảng trang bên ngoài thành cơ chế phân trang 3 cấp. Giả sử rằng bảng trang bên ngoài được tạo ra từ các trang có kích thước chuẩn (210 mục từ, hay 212 bytes); một không gian địa chỉ 64 bit vẫn có kích thước rất lớn:

Trang bên ngoài cấp 2	Trang bên ngoài	Trang bên trong	Độ dời
P1	P2	P3	D
32	10	10	12

Bảng trang bên ngoài vẫn lớn 232.

Bước tiếp theo sẽ là cơ chế phân trang cấp bốn, ở đây bảng trang bên ngoài cấp hai cũng được phân trang. Kiến trúc SPARC (với 32-bit đánh địa chỉ) hỗ trợ cơ chế phân trang cấp ba, trái lại kiến trúc Motorola 68030 32-bit hỗ trợ cơ chế phân trang bốn cấp.

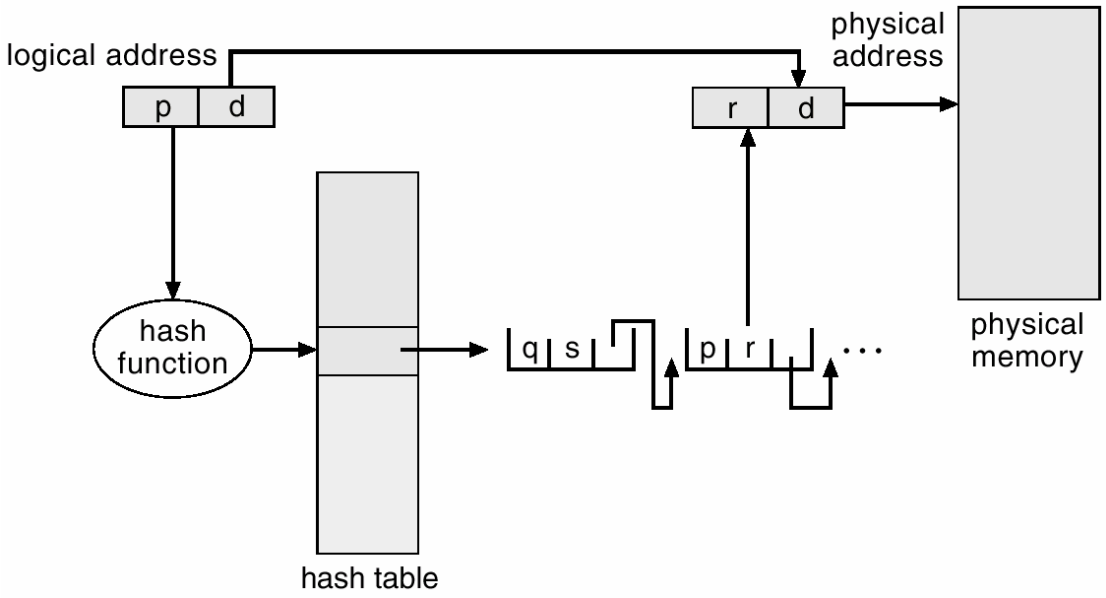
Tuy nhiên, đối với kiến trúc 64-bit, các bảng trang phân cấp thường được xem xét là không phù hợp. Thí dụ, UltraSPARC 64-bit sẽ yêu cầu phân trang bảy cấp – một số truy xuất bộ nhớ không được phép để dịch mỗi địa chỉ luận lý.

.VI.1.4.2 Bảng trang được băm

Một tiếp cận thông thường cho việc quản lý không gian địa chỉ lớn hơn 32-bit là dùng **bảng trang được băm** (hashed page table), với giá trị băm là số trang ảo. Mỗi mục từ trong bảng trang chứa một danh sách liên kết của các phần tử. Danh sách này băm tới cùng vị trí (để quản lý đụng độ). Mỗi phần tử chứa ba trường: (a) số trang ảo, (b) giá trị khung trang được ánh xạ và con trỏ chỉ tới phần tử kế tiếp trong danh sách liên kết.

Giải thuật thực hiện như sau: số trang ảo trong địa chỉ ảo được băm tới bảng băm. Số trang ảo được so sánh tới trường (a) trong phần tử đầu tiên của danh sách liên kết. Nếu có phần tử trùng khớp, khung trang tương ứng (trường (b) được dùng để hình thành địa chỉ vật lý mong muốn). Nếu không có phần tử nào trùng khớp, các mục từ tiếp theo trong danh sách liên kết được tìm kiếm số trang ảo trùng khớp. Cơ chế này được hiển thị trong hình VII-20 dưới đây:

Một biến thể đối với cơ chế này cho không gian địa chỉ 64-bit được đề nghị. Bảng trang được nhóm (Clustered page tables) tương tự như bảng băm ngoại trừ mỗi mục từ trong bảng băm tham chiếu tới nhiều trang (chẳng hạn như 16) hơn là một trang. Do đó, mục từ bảng trang đơn có thể lưu những ánh xạ cho nhiều khung trang vật lý. Bảng trang được nhóm đặc biệt có ích cho không gian địa chỉ rời nhau (**spare**), ở đó các tham chiếu bộ nhớ là không liên tục và tập hợp khắp không gian bộ nhớ.



Hình 0-20 Bảng trang được băm

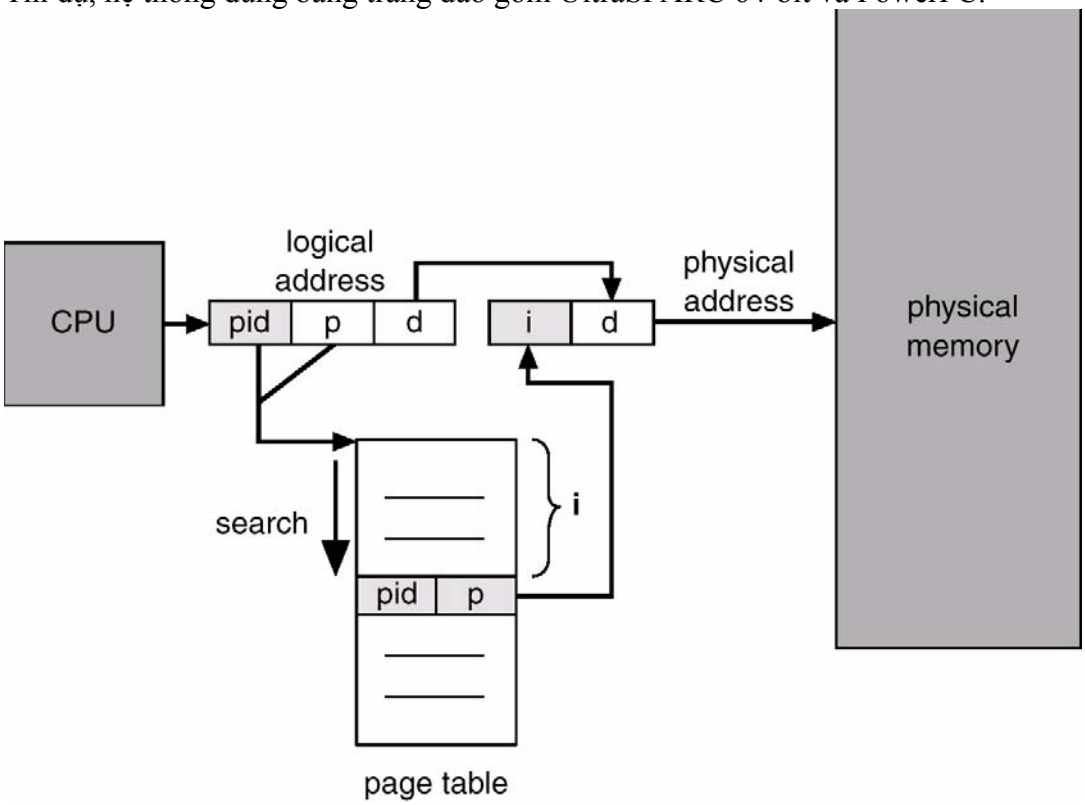
.VI.1.4.3 Bảng trang đảo

Thông thường, mỗi quá trình có một trang gán liền với nó. Bảng trang có một mục từ cho mỗi trang mà quá trình đó đang sử dụng (hay một khe cho mỗi địa chỉ ảo, không phụ thuộc tính hợp lệ sau đó). Biểu diễn bảng trang này là biểu diễn tự nhiên vì tham chiếu quá trình phân trang thông qua các địa chỉ ảo của trang. Sau đó, hệ điều hành phải dịch tham chiếu này vào một địa chỉ bộ nhớ vật lý. Vì bảng này được sắp xếp bởi địa chỉ ảo, hệ điều hành có thể tính toán nơi trong bảng mà mục từ địa chỉ vật lý được nối kết tới và sử dụng giá trị đó trực tiếp. Một trong những khó khăn của phương pháp này là mỗi bảng trang có thể chứa hàng triệu mục từ. Các bảng này có

thể tiêu tốn lượng lớn bộ nhớ vật lý, được yêu cầu chỉ để giữ vết của bộ nhớ vật lý khác đang được sử dụng như thế nào.

Để giải quyết vấn đề này chúng ta có thể sử dụng một **bảng trang đảo** (inverted page table). Bảng trang đảo có một mục từ cho mỗi trang thật (hay khung) của bộ nhớ. Mỗi mục từ chứa địa chỉ ảo của trang được lưu trong vị trí bộ nhớ thật đó, với thông tin về quá trình sở hữu trang đó. Do đó, chỉ một bảng trang trong hệ thống và nó chỉ có một mục từ cho mỗi trang của bộ nhớ vật lý. Hình VII-21 dưới đây hiển thị hoạt động của bảng trang đảo.

So sánh nó với hình VII-6, mô tả hoạt động của một bảng trang chuẩn. Vì chỉ một bảng trang trong hệ thống còn có nhiều không gian địa chỉ khác ánh xạ bộ nhớ vật lý, nên các bảng trang đảo thường yêu cầu một định danh không gian địa chỉ được lưu trong mỗi mục từ của bảng trang. Lưu trữ định danh không gian địa chỉ đảm bảo rằng ánh xạ của trang luận lý cho một quá trình xác định tới khung trang vật lý tương ứng. Thí dụ, hệ thống dùng bảng trang đảo gồm UltraSPARC 64-bit và PowerPC.



Hình 0-21 Bảng trang đảo

Để hiển thị phương pháp này, chúng ta mô tả một ấn bản được đơn giản hoá cài đặt bảng trang đảo dùng trong IBM RT. Mỗi địa chỉ ảo trong hệ thống chứa bộ ba: <process-id, page-number, offset>.

Mỗi mục từ bảng trang đảo là một cặp <process-id, page-number>, ở đây process-id đảm bảo vai trò định danh không gian địa chỉ. Khi một tham chiếu bộ nhớ xảy ra, một phần của địa chỉ ảo, gồm <process-id, page-number>, được hiện diện trong hệ thống bộ nhớ. Sau đó, bảng trang đảo được tìm kiếm sự trùng khớp. Nếu sự trùng khớp được tìm thấy tại mục từ i thì địa chỉ vật lý <i, offset> được tạo ra. Nếu không tìm thấy thì một truy xuất địa chỉ không hợp lệ được cố gắng thực hiện.

Mặc dù cơ chế này giảm lượng bộ nhớ được yêu cầu để lưu mỗi bảng trang, nhưng nó gia tăng lượng thời gian cần cho việc tìm kiếm bảng khi có một tham chiếu

xảy ra. Vì bảng trang đảo được lưu bởi địa chỉ vật lý nhưng tìm kiếm xảy ra trên địa chỉ ảo, toàn bộ bảng trang có thể cần được tìm kiếm sự trùng khớp. Sự tìm kiếm này có thể mất thời gian quá dài. Để làm giảm vấn đề này, chúng ta sử dụng một bảng băm được mô tả trong hình dưới đây để giới hạn việc tìm kiếm. Dĩ nhiên, mỗi truy xuất tới bảng băm thêm một tham chiếu tới thủ tục, để mà một tham chiếu bộ nhớ ảo yêu cầu ít nhất hai thao tác đọc bộ nhớ thật: một cho mục từ bảng băm và một cho bảng trang. Để cải tiến năng lực thực hiện, TLB được tìm kiếm đầu tiên, trước khi bảng băm được tra cứu.

VI.1.5 Trang được chia sẻ

Một thuận lợi khác của phân trang là khả năng chia sẻ mã chung. Việc xem xét này đặc biệt quan trọng trong môi trường chia thời. Xét một hệ thống hỗ trợ 40 người dùng, mỗi người dùng thực thi một trình soạn thảo văn bản. Nếu trình soạn thảo văn bản chứa 150 KB mã và 50 KB dữ liệu, chúng ta sẽ cần 8000 KB để hỗ trợ 40 người dùng. Tuy nhiên, nếu mã là mã tái sử dụng (reentrant code), nó có thể được chia sẻ như được hiển thị trong hình VII-22. Ở đây chúng ta thấy một bộ soạn thảo ba trang-mỗi trang có kích thước 50 KB; kích thước trang lớn được dùng để đơn giản hoá hình này-đang được chia sẻ giữa ba quá trình. Mỗi quá trình có trang dữ liệu riêng của nó.

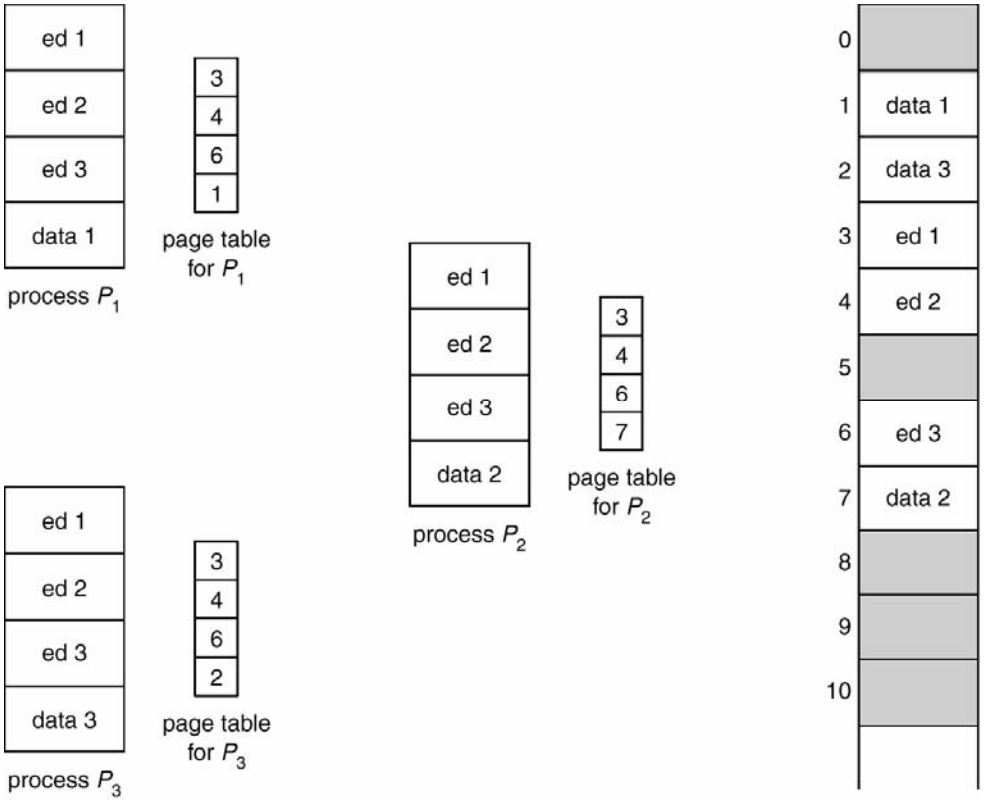
Mã tái sử dụng (hay thuần mã-pure code) là mã không thay đổi bởi chính nó. Nếu mã là tái sử dụng thì nó không bao giờ thay đổi trong quá trình thực thi. Do đó, hai hay nhiều quá trình có thể thực thi cùng mã tại cùng thời điểm. Mỗi quá trình có bản sao thanh ghi của chính nó và lưu trữ dữ liệu để quản lý dữ liệu cho việc thực thi của quá trình. Dĩ nhiên, dữ liệu cho hai quá trình khác nhau sẽ khác nhau cho mỗi quá trình.

Chỉ một bản sao của bộ soạn thảo cần được giữ trong bộ nhớ vật lý. Mỗi bảng trang của người dùng ánh xạ tới cùng bản sao vật lý của bộ soạn thảo nhưng các trang dữ liệu được ánh xạ tới các khung khác nhau. Do đó, để hỗ trợ 40 người dùng, chúng ta cần chỉ một bản sao của bộ soạn thảo (150 KB) cộng với 40 bản sao của 50 KB không gian dữ liệu trên một người dùng. Bây giờ toàn bộ không gian được yêu cầu là 2150 KB thay vì 8000 KB-một tiết kiệm lớn.

Những chương trình được dùng nhiều khác cũng có thể được chia sẻ - trình biên dịch, hệ thống cửa sổ, thư viện thời điểm thực thi, hệ cơ sở dữ liệu,…Để có thể chia sẻ, mã phải được tái sử dụng. Tính tự nhiên chỉ đọc của mã được chia sẻ sẽ không được phó mặc cho tính đúng đắn của mã; hệ điều hành nên tuân theo thuộc tính này. Chia sẻ bộ nhớ giữa các quá trình trên hệ điều hành tương tự chia sẻ không gian địa chỉ của một tác vụ bởi luồng. Ngoài ra, bộ nhớ được chia sẻ như một phương pháp giao tiếp liên quá trình. Một số hệ điều hành cài đặt bộ nhớ được chia sẻ dùng các trang được chia sẻ.

Hệ điều hành dùng bảng trang bên trong gặp khó khăn khi cài đặt bộ nhớ được chia sẻ. Bộ nhớ được chia sẻ thường được cài đặt như nhiều địa chỉ ảo (một địa chỉ cho mỗi quá trình chia sẻ bộ nhớ) mà chúng được ánh xạ tới một địa chỉ vật lý. Tuy nhiên, phương pháp chuẩn này không thể được dùng khi có chỉ một mục từ trang ảo cho mỗi trang vật lý vì thế một trang vật lý không thể có hai (hay nhiều) địa chỉ ảo được chia sẻ.

Tổ chức bộ nhớ dựa theo trang cung cấp nhiều lợi điểm khác để cho phép nhiều quá trình chia sẻ cùng trang vật lý.



Hình 0-22 chia sẻ mã trong môi trường phân trang

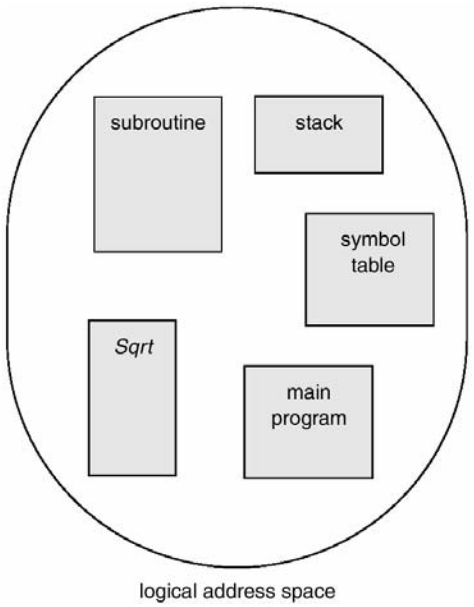
VI.2 Phân đoạn

Một khía cạnh quan trọng của việc quản lý bộ nhớ mà trở nên không thể tránh với phân trang là ngăn cách tầm nhìn bộ nhớ của người dùng và bộ nhớ vật lý thật sự. Tầm nhìn bộ nhớ của người dùng không giống như bộ nhớ vật lý. Tầm nhìn người dùng được ánh xạ vào bộ nhớ vật lý. Việc ánh xạ cho phép sự khác nhau giữa bộ nhớ luận lý và bộ nhớ vật lý.

VI.2.1 Phương pháp cơ bản

Người dùng nghĩ bộ nhớ như mảng tuyến tính các byte, một số byte chứa chỉ thị lệnh, một số khác chứa dữ liệu hay không? Hầu hết mọi người nói không. Đúng hơn là, người dùng thích nhìn bộ nhớ như tập hợp các phân đoạn có kích thước thay đổi, và không cần xếp thứ tự giữa các phân đoạn (như hình VII-23).

Chúng ta nghĩ như thế nào về một chương trình khi chúng ta đang viết nó? Chúng ta nghĩ nó như một chương trình chính với một tập hợp các chương trình con, thủ tục, hàm, hay các module. Có thể có các cấu trúc dữ liệu khác nhau: bảng, mảng, ngăn xếp, biến,..Mỗi module hay thành phần dữ liệu này được tham chiếu bởi tên. Chúng ta nói "bảng danh biểu", "hàm sqrt", "chương trình chính" không quan tâm đến địa chỉ trong bộ nhớ mà những phần tử này chiếm. Chúng ta không quan tâm bảng danh biểu được lưu trữ trước hay sau hàm sqrt. Mỗi phân đoạn này có chiều dài thay đổi; thực chất chiều dài được định nghĩa bởi mục đích của phân đoạn trong chương trình. Các phần tử trong một phân đoạn được định nghĩa bởi độ dời của chúng từ điểm bắt đầu của phân đoạn: lệnh đầu tiên của chương trình, mục từ thứ mười bảy trong bảng danh biểu, chỉ thị thứ năm của hàm sqrt,…



Hình 0-23 Tầm nhìn chương trình của người dùng

Phân đoạn là một cơ chế quản lý bộ nhớ hỗ trợ tầm nhìn bộ nhớ của người dùng. Không gian địa chỉ luận lý là tập hợp các phân đoạn. Mỗi phân đoạn có tên và chiều dài. Các địa chỉ xác định tên phân đoạn và độ dời trong phân đoạn. Do đó, người dùng xác định mỗi địa chỉ bằng hai lượng: tên phân đoạn và độ dời. (tương phản cơ chế này với cơ chế phân trang, trong đó người dùng chỉ xác định một địa chỉ đơn, được chia bởi phần cứng thành số trang và độ dời, tất cả không thể nhìn thấy đối với người lập trình).

Để đơn giản việc cài đặt, các phân đoạn được đánh số và được tham chiếu tới bởi số phân đoạn, hơn là bởi tên phân đoạn. Do đó, địa chỉ luận lý chứa một bộ hai:

<số phân đoạn, độ dời>

Thông thường, chương trình người dùng được biên dịch, và trình biên dịch tự động tạo ra các phân đoạn phản ánh chương trình nhập. Một chương trình Pascal có thể tạo các phân đoạn riêng như sau:

1) Các biến toàn cục;
2) Ngăn xếp gọi thủ tục, để lưu trữ các tham số và trả về các địa chỉ;
3) Phần mã của mỗi thủ tục hay hàm;
4) Các biến cục bộ của mỗi thủ tục và hàm

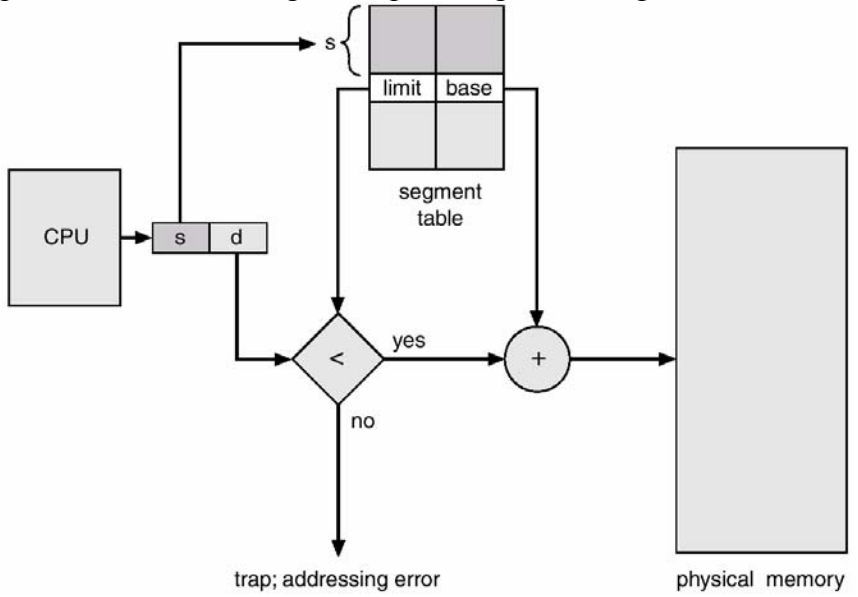
Một trình biên dịch có thể tạo một phân đoạn riêng cho mỗi khối chung. Các mảng có thể được gán các phân đoạn riêng. Bộ nạp có thể mang tất cả phân đoạn này và gán chúng số phân đoạn.

VI.2.2 Phần cứng

Mặc dù người dùng có thể tham chiếu tới các đối tượng trong chương trình bởi một địa chỉ hai chiều, bộ nhớ vật lý là chuỗi một chiều các byte. Do đó, chúng ta phải xác định việc cài đặt để ánh xạ địa chỉ hai chiều được định nghĩa bởi người dùng vào địa chỉ vật lý một chiều. Ánh xạ này được tác động bởi một bảng phân đoạn. Mỗi mục từ của bảng phân đoạn có một nền phân đoạn (segment base) và giới hạn phân đoạn

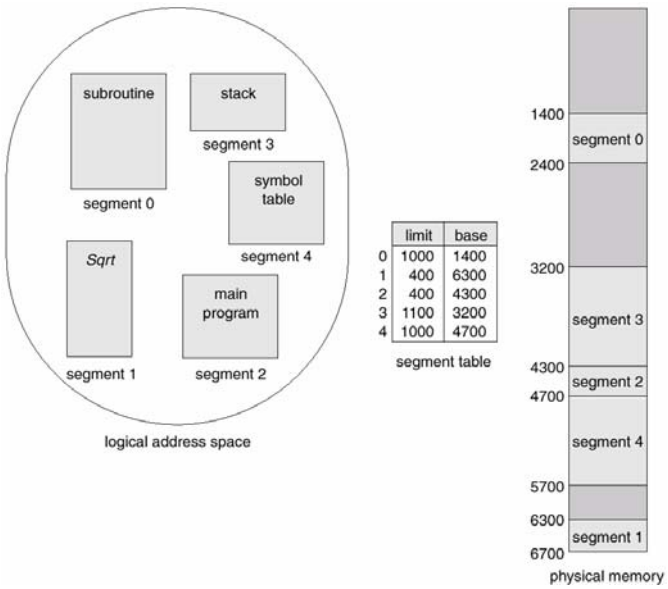
(segment limit). Nền phân đoạn chứa địa chỉ vật lý bắt đầu, nơi phân đoạn định vị trong bộ nhớ, ngược lại giới hạn phân đoạn xác định chiều dài của phân đoạn.

Sử dụng bảng phân đoạn được hiển thị như hình VII-24. Một địa chỉ luận lý có hai phần: số phân đoạn s và độ dời phân đoạn d. Số phân đoạn được dùng như chỉ mục trong bảng đoạn. Độ dời d của địa chỉ luận lý phải ở trong khoảng từ 0 tới giới hạn đoạn. Nếu không chúng ta sẽ trap tới hệ điều hành (địa chỉ vật lý vượt qua điểm cuối của phân đoạn). Nếu độ dời này là hợp lệ thì nó được cộng thêm giá trị nền của phân đoạn để tạo ra địa chỉ trong bộ nhớ vật lý của byte mong muốn. Do đó, bảng phân đoạn là một mảng của cặp thanh ghi nền và giới hạn.



Hình 0-24 Phần cứng phân đoạn

Xét trường hợp như hình VII-25. Chúng ta có năm phân đoạn được đánh số từ 0 đến 4. Các phân đoạn được lưu trong bộ nhớ vật lý như được hiển thị. Bảng phân đoạn có một mục từ riêng cho mỗi phân đoạn, cho địa chỉ bắt đầu của phân đoạn trong bộ nhớ vật lý (hay nền) và chiều dài của phân đoạn đó (hay giới hạn). Thí dụ, phân đoạn 2 dài 400 bytes và bắt đầu tại vị trí 4300. Do đó, một tham chiếu byte 53 của phân đoạn 2 được ánh xạ tới vị trí 4300 + 53 = 4353. Một tham chiếu tới phân đoạn 3, byte 852, được ánh xạ tới 3200 (giá trị nền của phân đoạn 3) +852=4052. Một tham chiếu tới byte 1222 của phân đoạn 0 dẫn đến một trap tới hệ điều hành, khi phân đoạn này chỉ dài 1000 bytes.

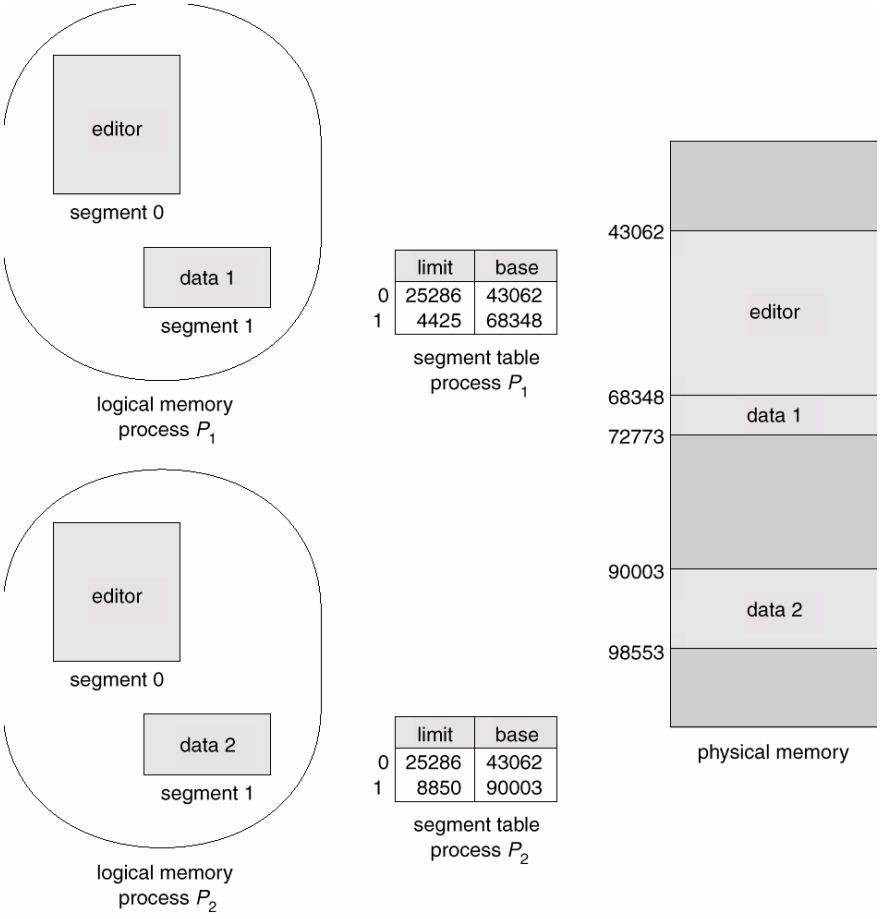


Hình 0-25 Thí dụ về phân đoạn

VI.2.3 Bảo vệ và chia sẻ

Lợi điểm đặc biệt của phân đoạn là sự gắn liền bảo vệ với các phân đoạn. Vì các phân đoạn biểu diễn một phần được định nghĩa của chương trình, tương tự như tất cả mục từ trong phân đoạn sẽ được dùng cùng một cách. Do đó, một số phân đoạn là chỉ thị, trong khi một số phân đoạn khác là dữ liệu. Trong một kiến trúc hiện đại, các chỉ thị không hiệu chỉnh chính nó vì thế các phân đoạn chỉ thị có thể được định nghĩa như chỉ đọc hay chỉ thực thi. Phần cứng ánh xạ bộ nhớ sẽ kiểm tra các bits bảo vệ được gắn với mỗi mục từ trong bảng phân đoạn để ngăn chặn các truy xuất không hợp lệ tới bộ nhớ, như cố gắng viết tới phân đoạn chỉ đọc hay sử dụng những phân đoạn chỉ đọc như dữ liệu. Bằng cách thay thế một mảng trong phân đoạn của chính nó, phần cứng quản lý bộ nhớ sẽ tự động kiểm tra các chỉ số của mảng là hợp lệ và không vượt ra ngoài giới hạn của mảng. Do đó, nhiều lỗi chương trình sẽ được phát hiện bởi phần cứng trước khi chúng có thể gây ra tác hại lớn.

Một lợi điểm khác liên quan đến chia sẻ mã hay dữ liệu. Mỗi quá trình có một bảng phân đoạn gắn với nó. Bộ phân phát dùng bảng phân đoạn này để định nghĩa phân đoạn phần cứng khi một quá trình được cấp CPU. Các phân đoạn được chia sẻ khi các mục từ trong bảng phân đoạn của hai quá trình khác nhau chỉ tới cùng một vị trí vật lý (như hình VII-26).



Hình 0-26 Chia sẻ các phân đoạn trong một hệ thống bộ nhớ được phân đoạn

Chia sẻ xảy ra tại cấp phân đoạn. Do đó, bất cứ thông tin có thể được chia sẻ nếu nó được định nghĩa là một phân đoạn. Một số phân đoạn có thể được chia sẻ vì thế một chương trình được hình thành từ nhiều phân đoạn có thể được chia sẻ.

Thí dụ, xét việc sử dụng một trình soạn thảo văn bản trong hệ thống chia thời. Trình soạn thảo hoàn chỉnh có thể rất lớn, được hình thành từ nhiều phân đoạn có thể được chia sẻ giữa tất cả người dùng, giới hạn địa chỉ vật lý được yêu cầu hỗ trợ các tác vụ soạn thảo. Thay vì n bản sao của trình soạn thảo, chúng ta chỉ cần một bản sao. Đối với mỗi người dùng, chúng ta vẫn cần các phân đoạn riêng, duy nhất để lưu các biến cục bộ. Dĩ nhiên, các phân đoạn này sẽ không được chia sẻ.

Chúng ta cũng có thể chia sẻ một số phần chương trình. Thí dụ, các gói chương trình con dùng chung có thể được chia sẻ giữa nhiều người dùng nếu chúng được định nghĩa như các phân đoạn chia sẻ, chỉ đọc. Thí dụ, hai chương trình Fortran có thể dùng cùng hàm Sqrt, nhưng chỉ một bản sao vật lý của hàm Sqrt được yêu cầu.

Mặc dù việc chia sẻ này có vẻ đơn giản, nhưng có những xem xét tinh tế. Điển hình, phân đoạn mã chứa các tham chiếu tới chính nó. Thí dụ, một lệnh nhảy (jump) có điều kiện thường có một địa chỉ chuyển gồm số phân đoạn và độ dời. Số phân đoạn của địa chỉ chuyển sẽ là số phân đoạn của phân đoạn mã. Nếu chúng ta cố gắng chia sẻ phân đoạn này, tất cả quá trình chia sẻ phải định nghĩa phân đoạn mã được chia sẻ để có cùng số phân đoạn.

Thí dụ, nếu chúng ta muốn chia sẻ hàm Sqrt và một quá trình muốn thực hiện nó phân đoạn 4 và một quá trình khác muốn thực hiện nó phân đoạn 17, hàm Sqrt nên tham chiếu tới chính nó như thế nào? Vì chỉ có một bản sao vật lý của Sqrt, nó phải được tham chiếu tới chính nó trong cùng cách cho cả hai người dùng-nó phải có một

số phân đoạn duy nhất. Khi số người dùng chia sẻ tăng do đó khó khăn trong việc tìm số phân đoạn có thể chấp nhận cũng tăng.

Các phân đoạn chỉ đọc không chứa con trỏ vật lý có thể được chia sẻ như số phân đoạn khác nhau, như các phân đoạn mã tham chiếu chính nó không trực tiếp. Thí dụ, các nhánh điều kiện xác định địa chỉ nhánh như một độ dời từ bộ đếm chương trình hiện hành hay quan hệ tới thanh ghi chứa số phân đoạn hiện hành nên cho phép mã tránh tham chiếu trực tiếp tới số phân đoạn hiện hành.

VI.2.4 Sự phân mãnh

Bộ định thời biểu dài phải tìm và cấp phát bộ nhớ cho tất cả các phân đoạn của chương trình người dùng. Trường hợp này tương tự như phân trang ngoại trừ các phân đoạn có chiều dài thay đổi; các trang có cùng kích thước. Do đó, với cơ chế phân khu có kích thước thay đổi, cấp phát bộ nhớ là một vấn đề cấp phát lưu trữ động, thường giải quyết với giải thuật best-fit hay first-fit.

Phân đoạn có thể gây ra sự phân mãnh, khi tất cả khối bộ nhớ trống là quá nhỏ để chứa một phân đoạn. Trong trường hợp này, quá trình có thể phải chờ cho đến khi nhiều bộ nhớ hơn (hay ít nhất một lỗ lớn hơn) trở nên sẳn dùng, hay cho tới khi việc hợp nhất các lỗ nhỏ để tạo một lỗ lớn hơn. Vì sự phân đoạn dùng giải thuật tái định vị động nên chúng ta có thể gom bộ nhớ bất cứ khi nào chúng ta muốn. Nếu bộ định thời biểu CPU phải chờ một quá trình vì vấn đề cấp phát bộ nhớ, nó có thể (hay không thể) bỏ qua hàng đợt CPU để tìm một quá trình nhỏ hơn, có độ ưu tiên thấp hơn để chạy.

Phân mãnh ngoài đối với cơ chế phân đoạn là vấn đề quan trọng như thế nào? Định thời biểu theo thuật ngữ dài với sự cô đặc sẽ giúp giải quyết vấn đề phân mãnh phải không? Câu trả lời phụ thuộc vào kích thước trung bình của phân đoạn. Ở mức độ cao nhất, chúng ta có thể định nghĩa mỗi quá trình là một phân đoạn. Tiếp cận này cắt giảm cơ chế phân khu có kích thước thay đổi. Ở cấp độ khác, mỗi byte có thể được đặt trong chính phân đoạn của nó và được cấp phát riêng. Sắp xếp này xoá đi việc phân mãnh bên ngoài; tuy nhiên mỗi byte cần một thanh ghi nền cho mỗi tái định vị của nó, gấp đôi bộ nhớ được dùng! Dĩ nhiên, bước luận lý tiếp theo-các phân đoạn nhỏ có kích thước cố định-là phân trang. Thông thường, nếu kích thước phân đoạn trung bình là nhỏ, phân mãnh ngoài cũng sẽ nhỏ. Vì cá nhân các phân đoạn là nhỏ hơn toàn bộ quá trình nên chúng có vẻ thích hợp hơn để đặt vào trong các khối bộ nhớ sẳn dùng.

VI.3 Phân đoạn với phân trang

Cả hai phân đoạn và phân trang có những lợi điểm và nhược điểm. Thật vậy, hai bộ vi xử lý phổ biến nhất hiện nay là: dòng Motorola 68000 được thiết kế dựa trên cơ sở không gian địa chỉ phẳng, ngược lại, họ Intel 80x86 và Petium dựa trên cơ sở phân đoạn. Cả hai là mô hình bộ nhớ hợp nhất hướng tới sự kết hợp của phân trang và phân đoạn. Chúng ta có thể kết hợp hai phương pháp để tận dụng lợi điểm của chúng. Sự kết hợp này được thể hiện tốt nhất bởi kết trúc của Intel 386.

Ấn bản IBM OS/2 32-bit là một hệ điều hành chạy trên đỉnh của kiến trúc Intel 386 (hay cao hơn). Intel 386 sử dụng phân đoạn với phân trang cho việc quản lý bộ nhớ. Số tối đa các phân đoạn trên quá trình là 16KB và mỗi phân đoạn có thể lớn tới 4GB. Kích thước trang là 4 KB. Chúng ta sẽ không cho một mô tả đầy đủ về cấu trúc quản lý bộ nhớ của Intel 386 trong giáo trình này. Thay vào đó, chúng ta sẽ trình bày các ý tưởng quan trọng.

Không gian địa chỉ luận lý của quá trình được chia thành hai phân khu. Phân khu thứ nhất chứa tới 8 KB các phân đoạn dành riêng cho quá trình đó. Phân khu thứ

hai chứa tới 8 KB các phân đoạn được chia sẻ giữa tất cả quá trình. Thông tin về phân khu thứ nhất được giữ trong bảng mô tả cục bộ (Local Descriptor Table-LDT), thông tin về phân khu thứ hai được giữ trong bảng mô tả toàn cục (Global Descriptor Table-GDL). Mỗi mục từ trong LDT và GDT chứa 8 bytes, với thông tin chi tiết về phân đoạn xác định gồm vị trí nền và chiều dài của phân đoạn đó.

Địa chỉ luận lý là một cặp (bộ chọn, độ dời), ở đây bộ chọn là một số 16-bit

S	G	P
13	1	2

Trong đó: s gán tới số phân đoạn, g hiển thị phân đoạn ở trong GDT hay LDT, và p giải quyết vấn đề bảo vệ. Độ dời là một số 32-bit xác định vị trí của byte (hay từ) trong phân đoạn.

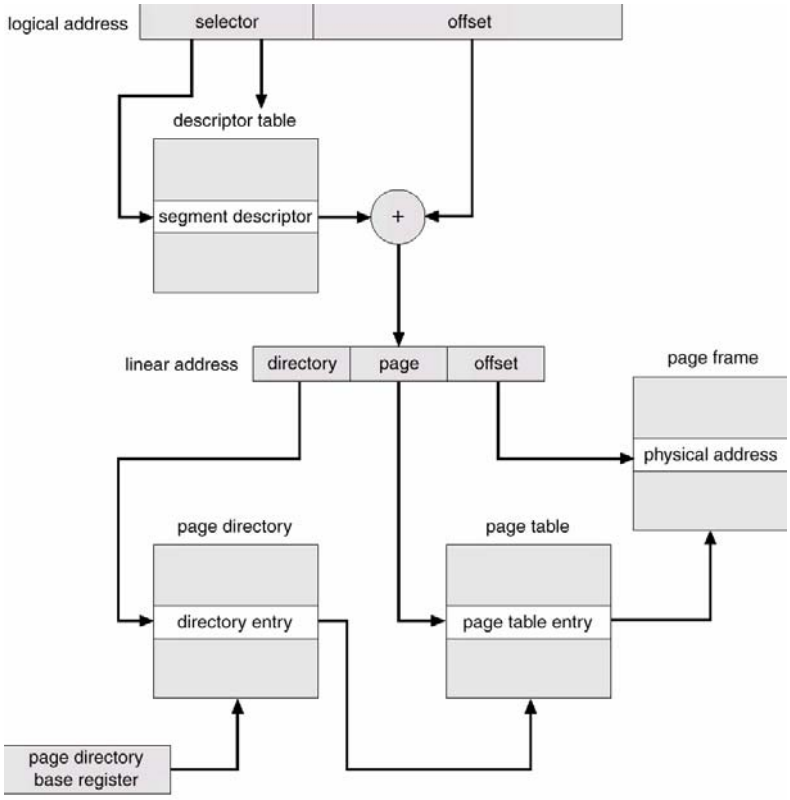
Máy này có 6 thanh ghi, cho phép 6 phân đoạn được xác định tại bất cứ thời điểm nào bởi một quá trình. Nó có 6 thanh ghi vi chương trình 8-byte để quản lý bộ mô tả tương ứng từ LDT hay GDT. Bộ lưu trữ này để Intel 386 tránh phải đọc bộ mô tả từ bộ nhớ cho mỗi lần tham chiếu bộ nhớ.

Địa chỉ vật lý trên 386 dài 32 bits và được hình thành như sau. Thanh ghi đoạn chỉ tới mục từ tương ứng trong LDT hay GDT. Thông tin nền và giới hạn về phân đoạn được dùng để phát sinh một địa chỉ tuyến tính. Đầu tiên, giới hạn được dùng để kiểm tra tính hợp lệ của địa chỉ. Nếu địa chỉ không hợp lệ, lỗi bộ nhớ được tạo ra, dẫn đến một trap tới hệ điều hành. Nếu nó là hợp lệ, thì giá trị của độ dời được cộng vào giá trị của nền, dẫn đến địa chỉ tuyến tính 32-bit. Sau đó, địa chỉ này được dịch thành địa chỉ vật lý.

Như được nêu trước đó, mỗi phân đoạn được phân trang và mỗi trang có kích thước 4 KB. Do đó, bảng trang có thể chứa tới 1 triệu mục từ. Vì mỗi mục từ chứa 4 bytes nên mỗi quá trình có thể cần 4 MB không gian địa chỉ vật lý cho một bảng trang. Rõ ràng, chúng ta không muốn cấp phát bảng trang liên tục trong bộ nhớ. Giải pháp này được thông qua trong Intel 386 để dùng cơ chế phân trang 2 cấp. Địa chỉ tuyến tính được chia thành số trang chứa 20 bits, và độ dời trang chứa 12 bits. Vì chúng ta phân trang bảng trang, số trang được chia nhỏ thành con trỏ thư mục trang 10-bit và con trỏ bảng trang 10-bit. Địa chỉ luận lý như sau:

P1	P2	D
10	10	12

Cơ chế dịch địa chỉ cho kiến trúc này tương tự như cơ chế được hiển thị trong hình VII-18. Dịch địa chỉ Intel được hiển thị chi tiết hơn trong hình VII-27dưới đây.



Hình 0-27 Dịch địa chỉ Intel 386

Để cải tiến tính hiệu quả của việc sử dụng bộ nhớ vật lý, bảng trang Intel 386 có thể được hoán vị tới đĩa. Trong trường hợp này, mỗi bit được dùng trong mục từ thư mục trang để hiển thị bảng mà mục từ đang chỉ tới ở trong bộ nhớ hay trên đĩa. Nếu bảng ở trên đĩa, hệ điều hành có thể dùng 31 bit còn lại để xác định vị trí đĩa của bảng; sau đó bảng có thể được mang vào bộ nhớ theo yêu cầu.

VII Tóm tắt

Các giải thuật quản lý bộ nhớ cho hệ điều hành đa chương trải dài từ tiếp cận hệ thống người dùng đơn tới phân đoạn được phân trang. Yếu tố quyết định lớn nhất của phương pháp được dùng trong hệ thống cụ thể là phần cứng được cải thiện. Mỗi địa chỉ bộ nhớ được được tạo ra bởi CPU phải được kiểm tra hợp lệ và có thể được ánh xạ tới một địa chỉ vật lý. Kiểm tra không thể được cài đặt (hữu hiệu) bằng phần mềm. Do đó, chúng ta bị ràng buộc bởi tính sẵn dùng phần cứng.

Các giải thuật quản lý bộ nhớ được thảo luận (cấp phát liên tục, phân trang, phân đoạn, và sự kết hợp của phân trang và phân đoạn) khác nhau trong nhiều khía cạnh. Trong so sánh các chiến lược quản lý bộ nhớ, chúng ta nên sử dụng các xem xét sau:

- **Hỗ trợ phần cứng**: thanh ghi nền hay cặp thanh ghi nền và thanh ghi giới hạn là đủ cho cơ chế phân khu đơn và đa, ngược lại phân trang và phân đoạn cần bảng ánh xạ để xác định ánh xạ địa chỉ.
- **Năng lực**: khi giải thuật quản lý bộ nhớ trở nên phức tạp hơn thì thời gian được yêu cầu để ánh xạ địa chỉ luận lý tới địa chỉ vật lý tăng. Đối với các hệ thống đơn giản, chúng ta chỉ cần so sánh hay cộng địa chỉ luận lý-các thao tác này phải nhanh. Phân trang và phân đoạn có thể nhanh như nếu

bảng được cài đặt trong các thanh ghi nhanh. Tuy nhiên, nếu bảng ở trong bộ nhớ thì về thực chất truy xuất bộ nhớ của người dùng có thể bị giảm. Một TLB có thể hạn chế việc giảm năng lực tới mức có thể chấp nhận.

- **Phân mãnh**: một hệ thống đa chương sẽ thực hiện hiệu quả hơn nếu nó có cấp độ đa chương cao hơn. Đối với một tập hợp quá trình được cho, chúng ta có thể tăng cấp độ đa chương chỉ bằng cách đóng gói nhiều quá trình hơn trong bộ nhớ. Để hoàn thành tác vụ này, chúng ta phải giảm sự lãng phí hay phân mãnh bộ nhớ. Các hệ thống với các đơn vị cấp phát có kích thước cố định, như cơ chế đơn phân khu và phân trang, gặp phải sự phân mãnh trong. Các hệ thống với đơn vị cấp phát có kích thước thay đổi như cơ chế đa phân khu và phân đoạn, gặp phải sự phân mãnh ngoài.
- **Tái định vị**: một giải pháp cho vấn đề phân mãnh ngoài là cô đặc. Cô đặc liên quan đến việc chuyển dịch một chương trình trong bộ nhớ không chú ý những thay đổi của chương trình. Xem xét này yêu cầu địa chỉ luận lý được tái định vị động tại thời điểm thực thi. Nếu địa chỉ được tái định vị chỉ tại thời điểm nạp, chúng ta không thể lưu trữ dạng cô đặc.
- **Hoán vị**: bất cứ giải thuật có thể có hoán vị được thêm tới nó. Tại những khoảng thời gian được xác định bởi hệ điều hành, thường được mô tả bởi các chính xác định thời, các quá trình được chép từ bộ nhớ chính tới vùng lưu trữ phụ và sau đó được chép trở lại tới bộ nhớ chính. Cơ chế này cho phép nhiều quá trình được chạy hơn là có thể đặt vào bộ nhớ tại cùng một thời điểm.
- **Chia sẻ**: một phương tiện khác để gia tăng cấp độ đa chương là chia sẻ mã và dữ liệu giữa các người dùng khác nhau. Thường việc chia sẻ yêu cầu phân trang hay phân đoạn được dùng, để cung cấp những gói thông tin nhỏ (các trang hay các đoạn) có thể được chia sẻ. Chia sẻ là một phương tiện chạy nhiều quá trình với lượng bộ nhớ giới hạn nhưng các chương trình và dữ liệu được chia sẻ phải được thiết kế cẩn thận.
- **Bảo vệ**: nếu phân trang hay phân đoạn được cung cấp, các phần khác nhau của chương trình người dùng có thể được khai báo chỉ thực thi, chỉ đọc, hay đọc-viết. Sự hạn chế này là cần thiết với mã và dữ liệu được chia sẻ và thường có ích trong bất cứ trường hợp nào để cung cấp việc kiểm tra tại thời gian thực thi cho các lỗi lập trình thông thường.