CHƯƠNG 6. QUẢN LÍ BỘ NHỚ ẢO

Trong chương trước, chúng ta thảo luận các chiến lược quản lý bộ nhớ được dùng trong hệ thống máy tính. Tất cả những chiến lược này có cùng mục đích: giữ nhiều tiến trình trong bộ nhớ cùng một lúc để cho phép đa chương. Tuy nhiên, chúng có khuynh hướng yêu cầu toàn bộ tiến trình ở trong bộ nhớ trước khi tiến trình có thể thực thi.

Bộ nhớ ảo là một kỹ thuật cho phép việc thực thi của tiến trình mà tiến trình có thể không hoàn toàn ở trong bộ nhớ. Một lợi điểm quan trọng của cơ chế này là các chương trình có thể lớn hơn bộ nhớ vật lý. Ngoài ra, bộ nhớ ảo phóng đại bộ nhớ chính thành bộ nhớ lô gic cực lớn khi được hiển thị bởi người dùng. Kỹ thuật này giải phóng người lập trình từ việc quan tâm đến giới hạn kích thước bộ nhớ. Bộ nhớ ảo cũng cho phép các tiến trình dễ dàng chia sẻ tập tin và không gian địa chỉ, cung cấp cơ chế hữu hiện cho việc tạo tiến trình.

Tuy nhiên, bộ nhớ ảo không dễ cài đặt và về thực chất có thể giảm năng lực nếu nó được dùng thiếu thận trọng. Trong chương này, chúng ta thảo luận bộ nhớ ảo trong dạng phân trang theo yêu cầu và xem xét độ phức tạp và chi phí.

I. Kiến thức cơ sở

Các giải thuật quản lý bộ nhớ trong chương trước là cần thiết vì một yêu cầu cơ bản: các chỉ thị đang được thực thi phải ở trong bộ nhớ vật lý. Tiếp cận đầu tiên để thoả mãn yêu cầu này đặt toàn bộ không gian địa chỉ lô gic trong bộ nhớ vật lý. Phủ lắp và nạp động có thể giúp làm giảm hạn chế này nhưng chúng thường yêu cầu sự đề phòng đặc biệt và công việc phụ thêm bởi người lập trình. Hạn chế này dường như cần thiết và phù hợp nhưng nó không may mắn vì nó giới hạn kích thước của một chương trình đối với kích thước bộ nhớ vật lý.

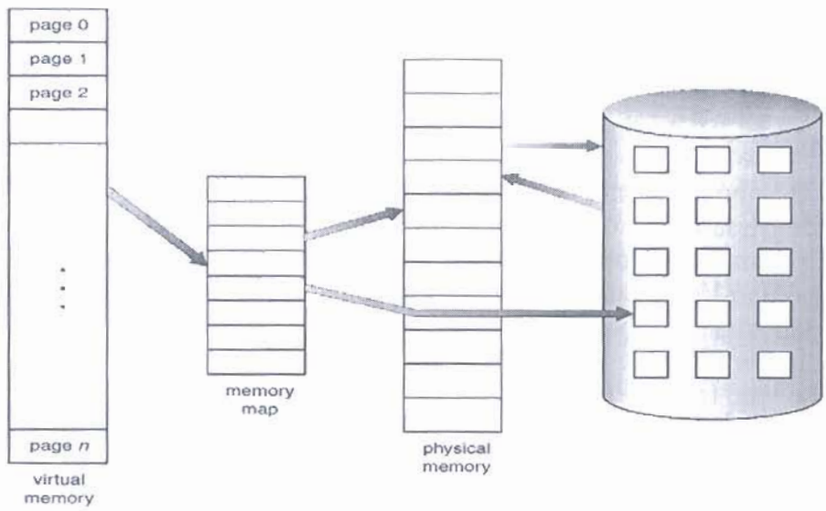
Thật vậy, xem xét các chương trình thực thi chúng ta nhận thấy rằng trong nhiều trường hợp toàn bộ chương trình là không cần thiết. Thậm chí trong những trường hợp toàn bộ chương trình được yêu cầu nhưng không phải tất cả chương trình được yêu cầu cùng một lúc.

Khả năng thực thi chương trình chỉ một phần chương trình ở trong bộ nhớ có nhiều lợi điểm:

- Chương trình sẽ không còn bị ràng buộc bởi không gian bộ nhớ vật lý sẵn có. Người dùng có thể viết chương trình có không gian địa chỉ ảo rất lớn, đơn giản hoá tác vụ lập trình.
- Vì mỗi chương trình người dùng có thể lấy ít hơn bộ nhớ vật lý nên nhiều chương trình hơn có thể được thực thi tại một thời điểm. Điều này giúp gia tăng việc sử dụng CPU và thông lượng nhưng không tăng thời gian đáp ứng.
- Yêu cầu ít nhập/xuất hơn để nạp hay hoán vị mỗi chương trình người dùng trong bộ nhớ vì thế mỗi chương trình người dùng sẽ chạy nhanh hơn. Do đó, chạy một chương trình mà nó không nằm hoàn toàn trong bộ nhớ có lợi cho cả người dùng và hệ thống.

Bộ nhớ ảo là sự tách biệt bộ nhớ lô gic từ bộ nhớ vật lý. Việc tách biệt này cho phép bộ nhớ ảo rất lớn được cung cấp cho người lập trình khi chỉ bộ nhớ vật lý nhỏ

hơn là sẵn dùng (hình 6.1). Bộ nhớ ảo thực hiện tác vụ lập trình dễ hơn nhiều vì người lập trình không cần lo lắng về lượng bộ nhớ vật lý sẵn có nữa hay về mã gì có thể được thay thế trong việc phủ lắp; thay vào đó, người lập trình có thể quan tâm vấn đề được lập trình. Trên những hệ thống hỗ trợ bộ nhớ ảo, việc phủ lắp hầu như biến mất.



Hình 6.1. Lưu đồ minh hoạ bộ nhớ ảo lơn hơn bộ nhớ vật lý

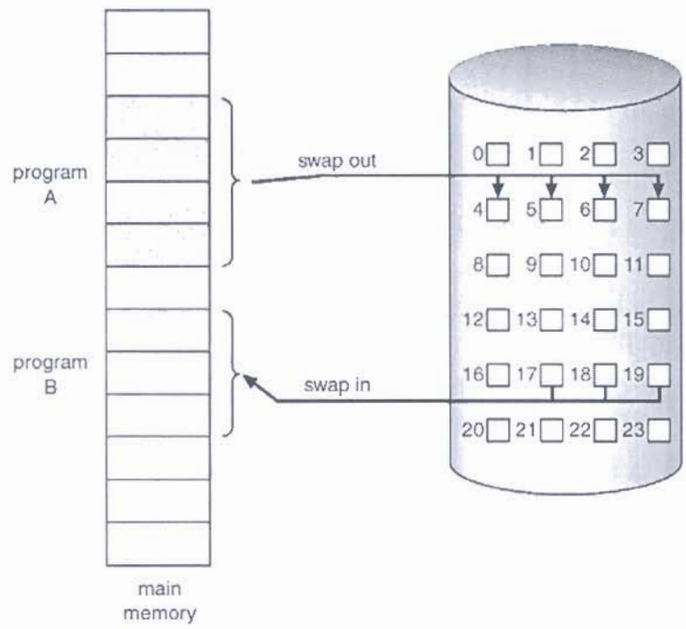
Thêm vào đó, việc tách biệt bộ nhớ lô gic từ bộ nhớ vật lý, bộ nhớ ảo cũng cho phép các tập tin và bộ nhớ được chia sẻ bởi những tiến trình khác nhau thông qua việc chia sẻ trang. Ngoài ra, chia sẻ trang cho phép cải tiến năng lực trong khi tạo tiến trình.

Bộ nhớ ảo thường được cài đặt bởi **phân trang theo yêu cầu** (demand paging). Nó cũng có thể được cài đặt trong cơ chế phân đoạn. Một vài hệ thống cung cấp cơ chế phân đoạn được phân trang. Trong cơ chế này các phân đoạn được chia thành các trang. Do đó, tầm nhìn người dùng là phân đoạn, nhưng hệ điều hành có thể cài đặt tầm nhìn này với cơ chế phân trang theo yêu cầu. Phân đoạn theo yêu cầu cũng có thể được dùng để cung cấp bộ nhớ ảo. Các hệ thống máy tính của Burrough dùng phân đoạn theo yêu cầu. Tuy nhiên, các giải thuật thay thế đoạn phức tạp hơn các giải thuật thay thế trang vì các đoạn có kích thước thay đổi. Chúng ta không đề cập phân đoạn theo yêu cầu trong giáo trình này.

II. Phân trang theo yêu cầu

Một hệ thống phân trang theo yêu cầu tương tự một hệ thống phân trang với hoán chuyển (hình 6.2). Các tiến trình định vị trong bộ nhớ phụ (thường là đĩa). Khi chúng ta muốn thực thi một tiến trình, chúng ta hoán chuyển nó vào bộ nhớ. Tuy nhiên, thay vì hoán chuyển toàn bộ tiến trình ở trong bộ nhớ, chúng ta dùng một **bộ hoán chuyển lười** (lazy swapper). Bộ hoán chuyển lười không bao giờ hoán chuyển một trang vào trong bộ nhớ trừ khi trang đó sẽ được yêu cầu. Vì bây giờ chúng ta xem một tiến trình như một chuỗi các trang hơn là một không gian địa chỉ liên tục có kích thước lớn, nên dùng hoán chuyển là không phù hợp về kỹ thuật. Một bộ hoán chuyển thao tác toàn bộ tiến trình, ngược lại một **bộ phân trang** (pager) được quan tâm với các trang riêng rẻ của một tiến trình.

Do đó, chúng ta dùng bộ phân trang (hơn là bộ hoán chuyển) trong nối kết với phân trang theo yêu cầu.



Hình 6.2. Chuyển bộ nhớ được phân trang tới không gian đĩa liên tục

II.1 Các khái niệm cơ bản

Với cơ chế này, chúng ta cần một số dạng phần cứng hỗ trợ để phân biệt giữa các trang ở trong bộ nhớ và các trang ở trên đĩa. Cơ chế bit hợp lệ-không hợp lệ có thể được dùng cho mục đích này. Tuy nhiên, thời điểm này khi bit được đặt "hợp lệ", giá trị này hiển thị rằng trang được tham chiếu tới là hợp lệ và ở đang trong bộ nhớ. Nếu một bit được đặt "không hợp lệ", giá trị này hiển thị rằng trang không hợp lệ (nghĩa là trang không ở trong không gian địa chỉ của tiến trình) hoặc hợp lệ nhưng hiện đang ở trên đĩa. Mục từ bảng trang cho trang không ở trong bộ nhớ đơn giản được đánh dấu không hợp lệ, hay chứa địa chỉ của trang trên đĩa. Trường hợp này được mô tả trong hình 6.3.

Chú ý rằng, đánh dấu một trang là "không hợp lệ" sẽ không có tác dụng nếu tiến trình không bao giờ truy xuất trang đó. Do đó, nếu chúng ta đoán đúng và tất cả những trang thật sự cần đều ở trong bộ nhớ, tiến trình sẽ chạy chính xác như khi chúng ta mang tất cả trang vào bộ nhớ. Trong khi tiến trình thực thi và truy xuất trang đang định vị trong bộ nhớ, việc thực thi xử lý bình thường.

Nhưng điều gì xảy ra nếu tiến trình cố gắng truy xuất trang mà trang đó không được mang vào bộ nhớ? Truy xuất một trang được đánh dấu là "không hợp lệ" gây ra một **trap lỗi trang** (page-fault trap). Phần cứng phân trang, dịch địa chỉ thông qua bảng trang, sẽ thông báo rằng bit không hợp lệ được đặt, gây ra một trap tới hệ điều hành. Trap này là kết quả lỗi của hệ điều hành mang trang được mong muốn vào bộ nhớ (trong một cố gắng tối thiểu chi phí chuyển đĩa và yêu cầu bộ nhớ) hơn là lỗi địa chỉ không hợp lệ như kết quả của việc cố gắng dùng một địa chỉ bộ nhớ không hợp lệ (như một ký hiệu mảng không hợp lệ). Do đó, chúng ta phải sửa trường hợp sơ xuất này. Thủ tục cho việc quản lý lỗi trang này là không phức tạp (hình 6.4).

1) Chúng ta kiểm tra bảng bên trong (thường được giữ với khối điều khiển tiến trình) cho tiến trình này, để xác định tham chiếu là truy xuất bộ nhớ hợp lệ hay không hợp lệ.

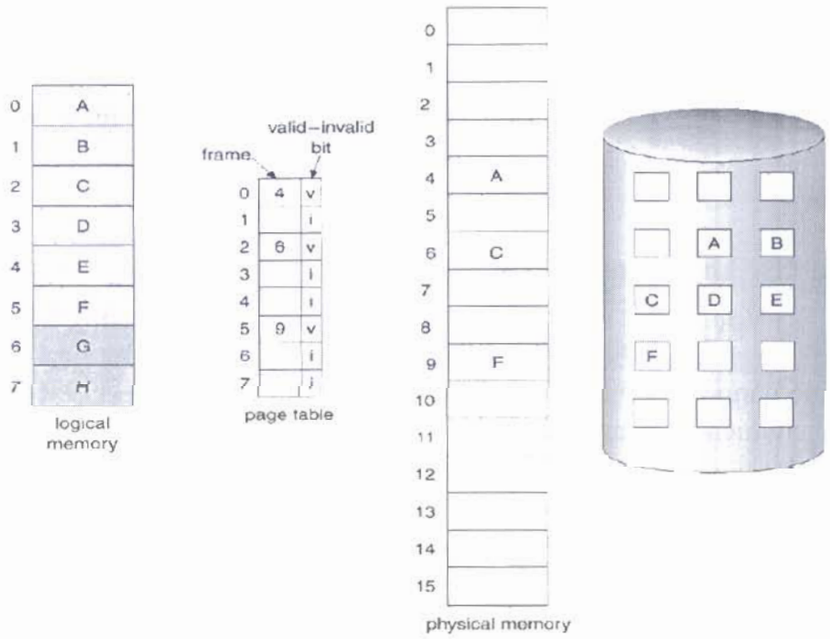
2) Nếu tham chiếu là không hợp lệ, chúng ta kết thúc tiến trình. Nếu nó là hợp lệ, nhưng chúng ta chưa mang trang đó vào bộ nhớ, bây giờ chúng ta mang trang đó vào.

3) Chúng ta tìm khung trống (thí dụ, bằng cách mang một trang từ danh sách khung trống).

4) Chúng ta lập thời biểu thao tác đĩa để đọc trang mong muốn vào khung trang vừa mới được cấp phát.

5) Khi đọc đĩa hoàn thành, chúng ta sửa đổi bảng bên trong với tiến trình và bảng trang để biểu thị rằng trang bây giờ ở trong bộ nhớ.

6) Chúng ta khởi động lại chỉ thị mà nó bị ngắt bởi trap địa chỉ không hợp lệ. Bây giờ tiến trình có thể truy xuất trang ở trong bộ nhớ.

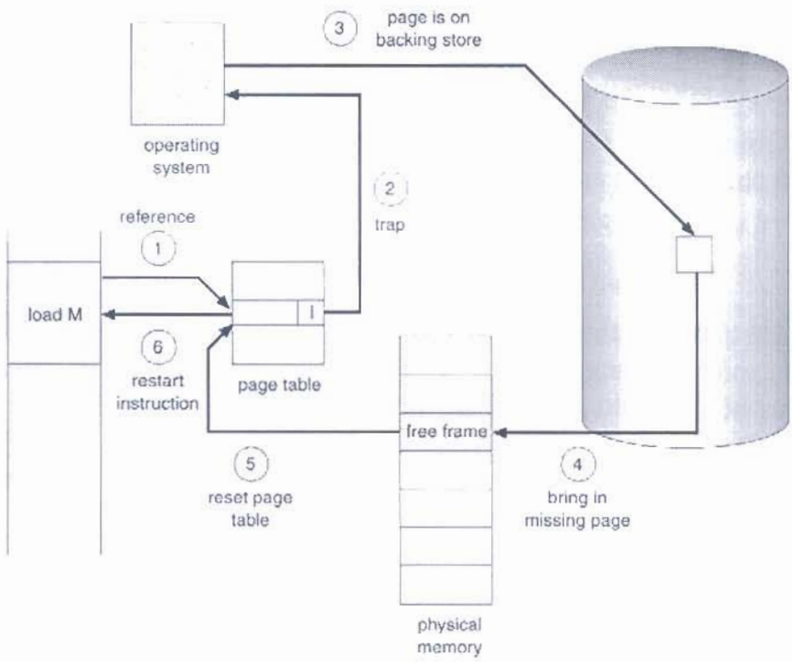


Hình 6.3. Bảng trang khi một số trang không ở trong bộ nhớ chính

Vì chúng ta lưu trạng thái (thanh ghi, mã điều kiện, bộ đếm chỉ thị lệnh) của tiến trình bị ngắt khi lỗi trang xảy ra, nên chúng ta có thể khởi động lại tiến trình chính xác nơi và trạng thái, ngoại trừ trang mong muốn hiện ở trong bộ nhớ và có thể truy xuất. Trong cách này, chúng ta có thể thực thi một tiến trình mặc dù các phần của nó chưa ở trong bộ nhớ. Khi tiến trình cố gắng truy xuất các vị trí không ở trong bộ nhớ, phần cứng trap tới hệ điều hành (lỗi trang). Hệ điều hành đọc trang được yêu cầu vào bộ nhớ và khởi động lại tiến trình như thể trang luôn ở trong bộ nhớ.

Trong trường hợp xấu nhất, chúng ta bắt đầu thực thi một tiến trình với không trang nào ở trong bộ nhớ. Khi hệ điều hành đặt con trỏ chỉ thị lệnh tới chỉ thị đầu tiên của tiến trình. Tuy nhiên, chỉ thị này ở trên trang không nằm trong bộ nhớ, tiến trình lập tức báo lỗi đối với trang đó. Sau khi trang được mang vào trong bộ nhớ, tiến trình

tiếp tục thực thi, báo lỗi khi cần cho tới khi mỗi trang nó cần ở trong bộ nhớ. Tại thời điểm đó, nó có thể thực thi với không có lỗi nào nữa. Cơ chế này là **thuần phân trang yêu cầu** (pure demand paging): không bao giờ mang trang vào bộ nhớ cho tới khi nó được yêu cầu.



Hình 6.4. Các bước quản lý lỗi trang

Về lý thuyết, một số tiến trình có thể truy xuất nhiều trang mới của bộ nhớ với mỗi sự thực thi chỉ thị (một trang cho một chỉ thị và nhiều trang cho dữ liệu), có thể gây ra lỗi nhiều trang trên chỉ thị. Trường hợp này sẽ dẫn đến năng lực thực hiện hệ thống không thể chấp nhận. May thay, phân tích các tiến trình thực thi thể hiện rằng hành vi này là không hoàn toàn xảy ra. Các chương trình có khuynh hướng tham chiếu cục bộ dẫn đến năng lực phù hợp từ phân trang yêu cầu.

Phần cứng hỗ trợ phân trang theo yêu cầu là tương tự như phần cứng phân trang và hoán chuyển.

- **Bảng trang:** bảng này có khả năng đánh dấu mục từ không hợp lệ thông qua bit hợp lệ-không hợp lệ hay giá trị đặc biệt của các bit bảo vệ

- **Bộ nhớ phụ:** bộ nhớ này quản lý các trang không hiện diện trong bộ nhớ chính. Bộ nhớ phụ thường là đĩa tốc độ cao. Nó được xem như là thiết bị hoán chuyển và phần đĩa được dùng cho mục đích này được gọi là không gian hoán chuyển.

Ngoài sự hỗ trợ phần cứng này, phần mềm có thể xem xét được yêu cầu. Ràng buộc kiến trúc phải được áp đặt. Ràng buộc quan trọng được yêu cầu là có thể khởi động lại bất cứ chỉ thị nào sau khi lỗi trang. Trong hầu hết các trường hợp, yêu cầu này là dễ dàng thoả mãn. Lỗi trang có thể xảy ra tại bất cứ tham chiếu bộ nhớ nào. Nếu lỗi trang xảy ra trên việc lấy chỉ thị, chúng ta có thể khởi động lại bằng cách lấy lại chỉ thị. Nếu lỗi trang xảy ra trong khi chúng ta đang lấy một toán hạng, chúng ta phải lấy và giải mã lại chỉ thị, và sau đó lấy toán hạng.

II.2 Năng lực của phân trang theo yêu cầu

Phân trang theo yêu cầu có thể có một ảnh hưởng lớn trên năng lực của một hệ thống máy tính. Để thấy tại sao, chúng ta tính **thời gian truy xuất hiệu quả** (effective access time) cho bộ nhớ được phân trang theo yêu cầu. Đối với hầu hết các hệ thống máy tính, thời gian truy xuất bộ nhớ, được ký hiệu *ma,* nằm trong khoảng từ 10 đến 200 nano giây. Với điều kiện là chúng ta không có lỗi trang, thời gian truy xuất hiệu quả là bằng với thời gian truy xuất bộ nhớ. Tuy nhiên, nếu lỗi trang xảy ra, trước hết chúng ta phải đọc trang tương ứng từ đĩa và sau đó truy xuất từ mong muốn.

Gọi p là xác suất của lỗi trang ($0 \le p \le 1$). Chúng ta mong đợi p gần bằng 0; nghĩa là chỉ có một vài lỗi trang.

Thời gian truy xuất hiệu quả là:

(Thời gian truy xuất hiệu quả) = (1 – p) x *ma* + p x (thời gian lỗi trang)

Để tính toán thời gian truy xuất hiệu quả, chúng ta phải biết phải mất bao lâu để phục vụ một lỗi trang. Để duy trì ở mức độ chấp nhận được sự chậm trễ trong hoạt động của hệ thống do phân trang, cần phải duy trì tỷ lệ phát sinh lỗi trang thấp.

III. Thay thế trang

Thay thế trang thực hiện tiếp cận sau. Nếu không có khung trống, chúng ta tìm một khung hiện không được dùng và giải phóng nó. Khi chúng ta giải phóng một khung bằng cách viết nội dung của nó tới không gian hoán chuyển và thay đổi bảng trang (và các bảng trang khác) để hiển thị rằng trang không còn ở trong bộ nhớ (hình 6.5). Bây giờ chúng ta có thể dùng khung được giải phóng để quản lý trang cho quá trình bị lỗi. Chúng ta sửa đổi thủ tục phục vụ lỗi trang để chứa thay thế trang:

1) Tìm vị trí trang mong muốn trên đĩa

2) Tìm khung trang trống

a) Nếu có khung trống, dùng nó.

b) Nếu không có khung trống, dùng một giải thuật thay thế trang để chọn khung "nạn nhân"

c) Viết trang "nạn nhân" tới đĩa; thay đổi bảng trang và khung trang tương ứng.

3) Đọc trang mong muốn vào khung trang trống; thay đổi bảng trang và khung trang.

4) Khởi động lại tiến trình.

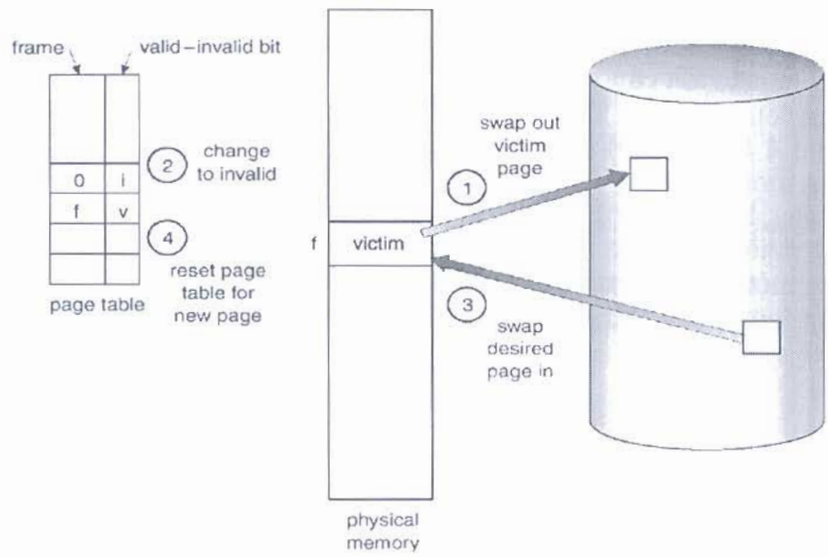
Chúng ta phải giải quyết hai vấn đề chính để cài đặt phân trang theo yêu cầu: Chúng ta phát triển giải thuật cấp phát khung và giải thuật thay thế trang. Nếu chúng ta có nhiều tiến trình trong bộ nhớ, chúng ta phải quyết định bao nhiêu khung cấp phát tới tiến trình. Ngoài ra, khi thay thế trang được yêu cầu, chúng ta phải chọn các khung để được thay thế. Thiết kế các giải thuật hợp lý để giải quyết vấn đề này là một tác vụ quan trọng vì nhập/xuất đĩa là rất đắt. Thậm chí một cải tiến nhỏ trong các phương pháp phân trang theo yêu cầu sinh ra một lượng lớn năng lực hệ thống.

Có nhiều giải thuật thay thế trang khác nhau. Mỗi hệ điều hành có thể có cơ chế thay thế của chính nó. Chúng ta chọn một giải thuật thay thế trang như thế nào? Thông thường, chúng ta muốn một giải thuật tỉ lệ lỗi trang nhỏ nhất.

Chúng ta đánh giá một giải thuật bằng cách chạy nó trên một chuỗi các tham

chiếu bộ nhớ cụ thể và tính số lượng lỗi trang. Chuỗi các tham chiếu bộ nhớ được gọi là chuỗi tham chiếu. Chúng ta có thể phát sinh chuỗi tham chiếu giả tạo (thí dụ, bằng bộ phát sinh số ngẫu nhiên). Chọn lựa sau đó tạo ra số lượng lớn dữ liệu (trên thứ tự 1triệu địa chỉ trên giây). Để làm giảm số lượng dữ liệu này, chúng ta có hai cách

Cách thứ nhất, đối với kích thước trang được cho (và kích thước trang thường được cố định bởi phần cứng hay hệ thống), chúng ta cần xét chỉ số trang hơn là toàn địa chỉ. Cách thứ hai, nếu chúng ta có một tham chiếu tới trang p, thì bất cứ những tham chiếu tức thì theo sau tới trang p sẽ không bao giờ gây lỗi trang. Trang p sẽ ở trong bộ nhớ sau khi tham chiếu đầu tiên; các tham chiếu theo sau tức thì sẽ không bị lỗi.



Hình 6.6. Mô tả thay thế trang

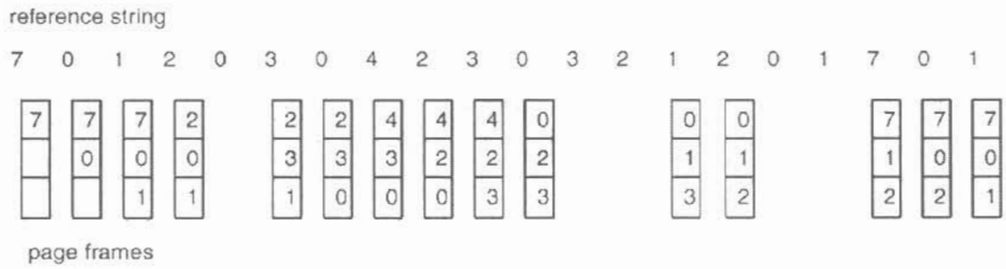
III.1. Thay thế trang FIFO

Giải thuật thay thế trang đơn giản nhất là giải thuật FIFO. Giải thuật này gắn mỗi trang với thời gian khi trang đó được mang vào trong bộ nhớ. Khi một trang phải được thay thế, trang cũ nhất sẽ được chọn. Chú ý rằng, nó không yêu cầu nghiêm ngặt để ghi thời gian khi trang được mang vào. Chúng ta có thể tạo một hàng đợi FIFO để quản lý tất cả trang trong bộ nhớ. Chúng ta thay thế trang tại đầu hàng đợi. Khi trang được mang vào bộ nhớ, chúng ta chèn nó vào đuôi của hàng đợi.

Cho một thí dụ về chuỗi tham chiếu, 3 khung của chúng ta ban đầu là rỗng. 3 tham chiếu đầu tiên (7, 0, 1) gây ra lỗi trang và được mang vào các khung rỗng này. Tham chiếu tiếp theo (2) thay thế trang 7, vì trang 7 được mang vào trước. Vì 0 là tham chiếu tiếp theo và 0 đã ở trong bộ nhớ rồi, chúng ta không có lỗi trang cho tham chiếu này. Tham chiếu đầu tiên tới 3 dẫn đến trang 0 đang được thay thế vì thế nó là trang đầu tiên của 3 trang trong bộ nhớ (0, 1, 2) để được mang vào. Bởi vì thay thế này, tham chiếu tiếp theo, tới 0, sẽ bị lỗi. Sau đó, trang 1 được thay thế bởi trang 0. Tiến trình này tiếp tục như được hiển thị trong hình 6.6. Mỗi khi một lỗi xảy ra, chúng ta hiển thị các trang ở trong 3 khung của chúng ta. Có 15 lỗi cả thảy.

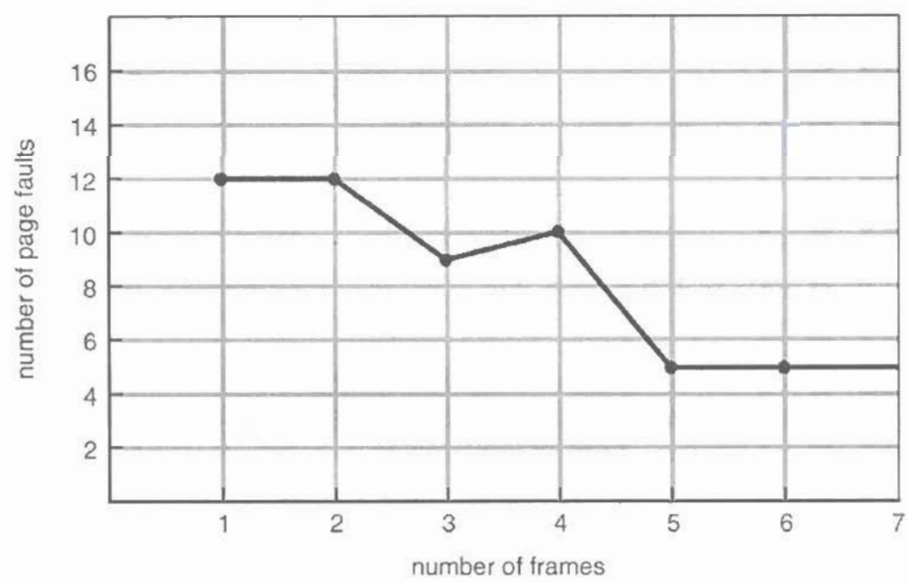
Giải thuật thay thế trang FIFO rất dễ hiểu và lập trình. Tuy nhiên, năng lực của nó không luôn tốt. Trang được cho để thay thế có thể là trang chứa nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi chuyển ra bộ nhớ phụ

sẽ nhanh chóng gây ra lỗi trang.



Hình 6.6. Giải thuật thay thế trang FIFO

Để hiển thị các vấn đề có thể phát sinh với giải thuật thay thế trang FIFO, chúng ta xem xét chuỗi tham chiếu sau: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 6. Hình 6.7 hiển thị đường cong lỗi trang khi so sánh với số khung sẵn dùng. Chúng ta chú ý rằng số lượng lỗi cho 4 khung (10) là lớn hơn số lượng lỗi cho 3 khung (9). Hầu hết các kết quả không mong đợi này được gọi là sự nghịch lý Belady; đối với một số giải thuật thay thế trang, tỉ lệ lỗi trang có thể tăng khi số lượng khung được cấp phát tăng. Chúng ta sẽ mong muốn rằng cho nhiều bộ nhớ hơn tới một tiến trình sẽ cải tiến năng lực của nó. Trong một vài nghiên cứu trước đây, các nhà điều tra đã kết luận rằng giả thuyết này không luôn đúng. Sự không bình thường của Belady được phát hiện như là một kết quả.



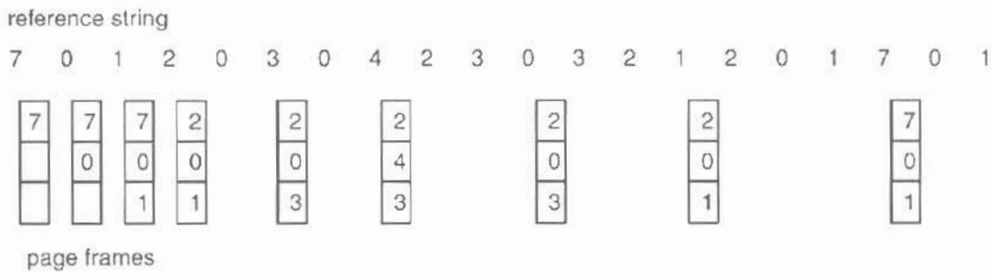
Hình 6.7. Đường cong lỗi trang cho thay thế FIFO trên chuỗi tham chiếu

III.2. Thay thế trang tối ưu hoá

Kết quả phát hiện sự nghịch lý của Belady là tìm ra một giải thuật thay thế trang tối ưu. Giải thuật thay thế trang tối ưu có tỉ lệ lỗi trang thấp nhất trong tất cả các giải thuật và sẽ không bao giờ gặp phải sự nghịch lý của Belady. Giải thuật như thế tồn tại và được gọi là OPT hay MIN. Nó đơn giản là: thay thế trang mà nó sẽ không được dùng cho một khoảng thời gian lâu nhất. Sử dụng giải thuật thay thế trang đảm bảo tỉ

lệ lỗi trang nhỏ nhất có thể cho một số lượng khung cố định. Thí dụ, trên một chuỗi tham chiếu mẫu, giải thuật thay thế trang tối ưu sẽ phát sinh 9 lỗi trang, như được hiển thị trong hình 6.8. 3 tham chiếu đầu tiên gây ra lỗi điền vào 3 khung trống. Tham chiếu tới trang 2 thay thế trang 7 vì 7 sẽ không được dùng cho tới khi tham chiếu 18, trái lại trang 0 sẽ được dùng tại 5 và trang 1 tại 14. Tham chiếu tới trang 3 thay thế trang 1 khi trang 1 sẽ là trang cuối cùng của 3 trang trong bộ nhớ được tham chiếu lần nữa. Với chỉ 9 lỗi trang, thay thế tối ưu là tốt hơn nhiều giải thuật FIFO, có 15 lỗi. (Nếu chúng ta bỏ qua 3 lỗi đầu mà tất cả giải thuật phải gặp thì thay thế tối ưu tốt gấp 2 lần thay thế FIFO) Thật vậy, không có giải thuật thay thế nào có thể xử lý chuỗi tham chiếu trong 3 khung với ít hơn 9 lỗi.

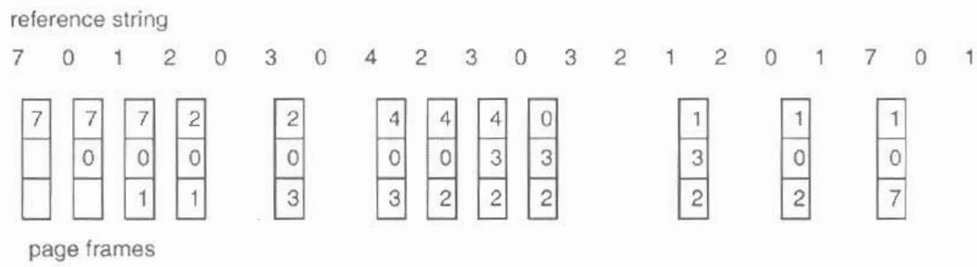
Tuy nhiên, giải thuật thay thế trang tối ưu là khó cài đặt vì nó yêu cầu kiến thức tương lai về chuỗi tham chiếu. Do đó, giải thuật tối ưu được dùng chủ yếu cho nghiên cứu so sánh. Thí dụ, nó có thể có ích để biết rằng, mặc dù một giải thuật không tối ưu nhưng nó nằm trong 12.3% của tối ưu là tệ, và trong 4.7% là trung bình.



Hình 6.8. Giải thuật thay thế trang tối ưu

III.3. Thay thế trang LRU

Nếu giải thuật tối ưu là không khả thi, có lẽ một xấp xỉ giải thuật tối ưu là có thể. Sự khác biệt chủ yếu giữa giải thuật FIFO và OPT là FIFO dùng thời gian khi trang được mang vào bộ nhớ; giải thuật OPT dùng thời gian khi trang được sử dụng. Nếu chúng ta sẽ dụng quá khứ gần đây như một xấp xỉ của tương lai gần thì chúng ta sẽ thay thế trang mà nó đã không được dùng cho khoảng thời gian lâu nhất (hình 6.9). Tiếp cận này là giải thuật ít được dùng gần đây nhất (least-recently-used (LRU)).

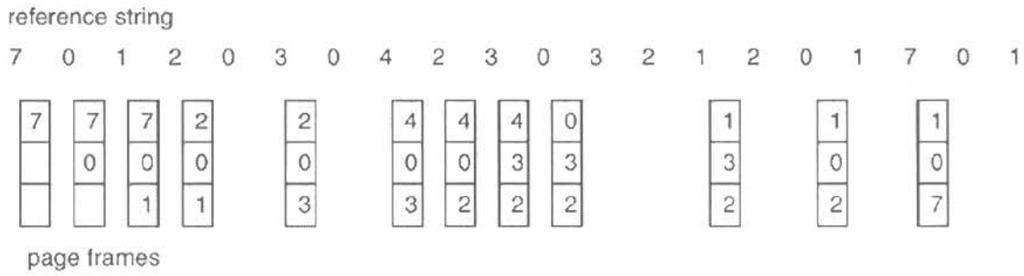


Hình 6.9. Giải thuật thay thế trang LRU

Thay thế trang LRU gắn với mỗi trang thời gian sử dụng cuối cùng của trang. Khi một trang phải được thay thế, LRU chọn trang không được dùng trong một khoảng thời gian lâu nhất. Chiến lược này là giải thuật thay thế trang tối ưu tìm kiếm lùi theo thời gian hơn là hướng tới. (gọi S_R là trình tự ngược của chuỗi tham chiếu S thì

tỉ lệ lỗi trang cho giải thuật OPT trên S là tương tự như tỉ lệ lỗi trang cho giải thuật OPT trên S_R. Tương tự, tỉ lệ lỗi trang đối với giải thuật LRU trên S là giống như tỉ lệ lỗi trang cho giải thuật LRU trên S_R)

Kết quả ứng dụng thay thế LRU đối với chuỗi tham chiếu điển hình được hiển thị trong hình 6.10. Giải thuật LRU sinh ra 12 lỗi. 5 lỗi đầu tiên là giống như thay thế tối ưu. Tuy nhiên, khi tham chiếu tới trang 4 xảy ra thay thế LRU thấy rằng 3 khung trong bộ nhớ, trang 2 được dùng gần đây nhất. Trang được dùng gần đây nhất là trang 0, và chỉ trước khi trang 3 được dùng. Do đó, giải thuật LRU thay thế trang 2, không biết rằng trang 2 để được dùng. Sau đó, khi nó gây lỗi trang 2, giải thuật LRU thay thế trang 3, của 3 trang trong bộ nhớ {0, 3, 4} trang 3 ít được sử dụng gần đây nhất. Mặc dù vấn đề này nhưng thay thế LRU với 12 lỗi vẫn tốt hơn thay thế FIFO với 16.



Hình 6.10. Giải thuật thay thế trang LRU

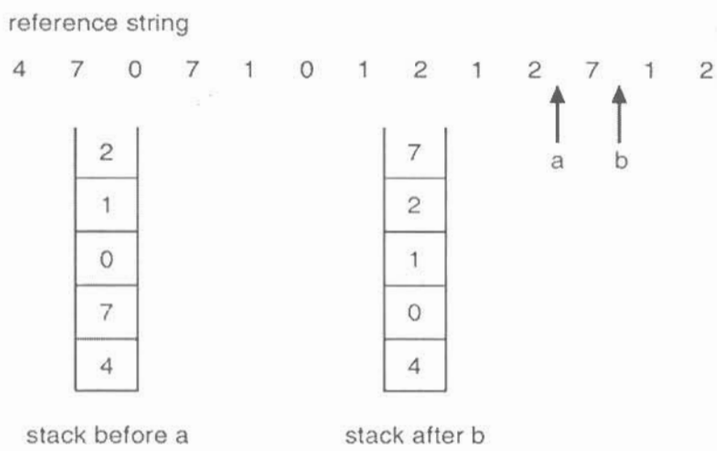
Chính sách LRU thường được dùng như giải thuật thay thế trang và được xem là tốt. Vấn đề chính là cách cài đặt thay thế LRU. Một giải thuật thay thế trang LRU có thể yêu cầu sự trợ giúp phần cứng. Vấn đề là xác định thứ tự cho các khung được định nghĩa bởi thời gian sử dụng gần nhất. Hai cách cài đặt khả thi là:

- **Bộ đếm:** trong trường hợp đơn giản nhất, chúng ta gắn mỗi mục từ bảng trang một trường số lần sử dụng và thêm CPU một đồng hồ lô gic hay bộ đếm. Đồng hồ được tăng cho mỗi tham chiếu bộ nhớ. Bất cứ khi nào một tham chiếu tới trang được thực hiện, các nội dung của thanh ghi đồng hồ được chép tới trường số lần sử dụng trong mục từ bảng trang cho trang đó. Trong cách này, chúng ta luôn có thời gian của tham chiếu cuối cùng tới mỗi trang. Chúng ta thay thế trang với giá trị số lần sử dụng nhỏ nhất. Cơ chế này yêu cầu tìm kiếm bảng trang để tìm ra trang LRU và viết tới bộ nhớ (tới trường thời gian dùng trong bảng trang) cho mỗi truy xuất bộ nhớ. Số lần cũng phải được duy trì khi các bảng trang bị thay đổi (do định thời CPU). Vượt quá giới hạn của đồng hồ phải được xem xét.

- **Ngăn xếp:** một tiếp cận khác để cài đặt thay thế LRU là giữ ngăn xếp số trang. Bất cứ khi nào một trang được tham chiếu, nó bị xoá từ ngăn xếp và đặt trên đỉnh. Trong cách này, đỉnh của ngăn xếp luôn là trang được dùng gần nhất và đáy là trang LRU (hình 6.11). Vì các mục từ phải được xoá từ giữa ngăn xếp, nó được cài đặt tốt nhất bởi một danh sách được liên kết kép với con trỏ đầu và đuôi. Xoá một trang và đặt nó trên đỉnh của ngăn xếp sau đó yêu cầu thay đổi 6 con trỏ trong trường hợp xấu nhất. Mỗi cập nhật là ít chi phí hơn nhưng không cần tìm một thay thế; con trỏ đuôi chỉ tới đáy của ngăn xếp là trang LRU. Tiếp cận này đặc biệt phù hợp cho cài đặt phần mềm hay vi mã của thay thế LRU.

Thay thế tối ưu hoá và LRU không gặp phải sự nghịch lý của Belady. Có một lớp giải thuật thay thế trang được gọi là giải thuật ngăn xếp, mà nó không bao giờ hiển thị sự nghịch lý Belady. Một giải thuật ngăn xếp là một giải thuật mà nó có thể được hiển thị rằng tập hợp trang trong bộ nhớ đối với n khung trang luôn là tập hợp con của tập hợp các trang mà nó ở trong bộ nhớ với (n + 1) khung. Đối với thay thế LRU, tập hợp trang trong bộ nhớ là n trang được tham chiếu gần đây nhất. Nếu số trang được gia tăng thì n trang này sẽ vẫn là những trang được tham chiếu gần đây nhất và vì thế sẽ vẫn ở trong bộ nhớ.

Chú ý rằng cài đặt LRU sẽ có thể không có sự trợ giúp phần cứng ngoại trừ thanh ghi TLB. Cập nhật các trường đồng hồ hay ngăn xếp phải được thực hiện cho mỗi tham chiếu bộ nhớ. Nếu chúng ta sử dụng ngắt cho mỗi tham chiếu bộ nhớ, cho phép phần mềm cập nhật cấu trúc dữ liệu thì nó sẽ làm chậm mỗi tham chiếu bộ nhớ gần 1 phần 10. Rất ít hệ thống có thể chịu cấp độ chi phí đó cho việc quản lý bộ nhớ.



Hình 6.11. Sử dụng ngăn xếp để ghi những tham chiếu trang gần nhất

III.4. Giải thuật thay thế trang xấp xỉ LRU

Rất ít hệ thống máy tính cung cấp đầy đủ hỗ trợ phần cứng cho thay thế trang LRU. Một số hệ thống không cung cấp bất cứ sự hỗ trợ phần cứng và giải thuật thay thế trang khác (như giải thuật FIFO) phải được dùng. Tuy nhiên, nhiều hệ thống cung cấp một vài hỗ trợ trong dạng 1 bit tham chiếu. Bit tham chiếu cho một trang được đặt bởi phần cứng, bất cứ khi nào trang đó được tham chiếu (đọc hay viết tới bất cứ bit nào trong trang). Các bit tham chiếu gắn liền với mỗi mục từ trong bảng trang.

Ban đầu, tất cả bit được xoá (tới 0) bởi hệ điều hành. Khi một tiến trình người dùng thực thi, bit được gán với mỗi trang được tham chiếu được đặt (tới 1) bởi phần cứng. Sau thời gian đó, chúng có thể xác định trang nào được dùng và trang nào không được dùng bằng cách xem xét các bit tham chiếu. Chúng ta không biết thứ tự sử dụng nhưng chúng ta biết trang nào được dùng và trang nào không được dùng. Thông tin thứ tự từng phần dẫn tới nhiều giải thuật thay thế trang xấp xỉ thay thế LRU.

III.4.1. Giải thuật các bit tham chiếu phụ

Chúng ta có thể nhận thêm thông tin thứ tự bằng cách ghi nhận các bit tham chiếu tại những khoảng thời gian đều đặn. Chúng ta có thể giữ một byte cho mỗi trang trong một bảng nằm trong bộ nhớ. Tại những khoảng thời gian đều đặn (mỗi 100 mili giây), một ngắt thời gian chuyển điều khiển tới hệ điều hành. Hệ điều hành chuyển bit

tham chiếu cho mỗi trang vào bit có trọng số lớn nhất của byte, dịch các bit còn lại sang phải 1 bit. Xoá bit có trọng số thấp nhất. Thanh ghi dịch 8 bit có thể chứa lịch sử của việc sử dụng trang đối với 8 lần gần nhất. Nếu thanh ghi dịch chứa 00000000, thì trang không được dùng cho 8 thời điểm; một trang được dùng ít nhất một lần mỗi thời điểm sẽ có giá trị thanh ghi dịch là 11111111.

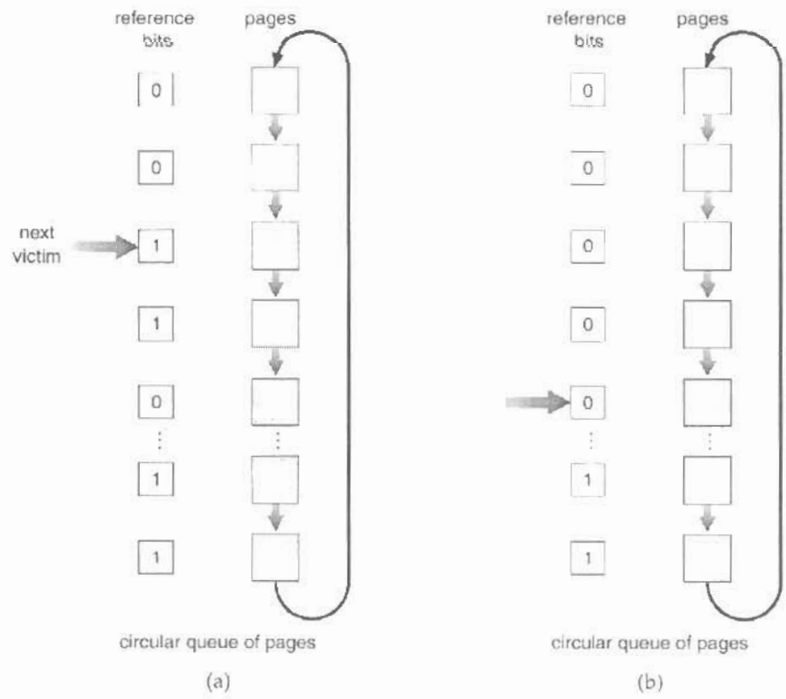
Một thanh ghi với giá trị thanh ghi lịch sử là 11000100 được dùng gần đây hơn một trang với 01110111. Nếu chúng ta thông dịch 8 bit này như số nguyên không dấu, trang với số thấp nhất là trang LRU và nó có thể được thay thế. Tuy nhiên, các số này không đảm bảo duy nhất, chúng ta thay thế tất cả trang với giá trị nhỏ nhất hay dùng FIFO để chọn giữa chúng.

Dĩ nhiên, số lượng bit lịch sử có thể khác nhau và có thể được chọn (phụ thuộc phần cứng sẵn có) để thực hiện cập nhật nhanh nhất có thể. Trong trường hợp cực độ, số có thể được giảm về 0, chỉ bit tham chiếu chính nó. Giải thuật này được gọi là **giải thuật thay thế trang cơ hội thứ hai** (second-chance page-replacement algorithm).

III.4.2. Giải thuật cơ hội thứ hai

Giải thuật thay thế trang cơ hội thứ hai cơ bản là giải thuật thay thế FIFO. Tuy nhiên, khi một trang được chọn, chúng ta xét bit tham chiếu của nó. Nếu giá trị bit này là 0, chúng ta xử lý để thay thế trang này. Tuy nhiên, nếu bit tham chiếu được đặt tới 1, chúng ta cho trang đó một cơ hội thứ hai và di chuyển để chọn trang FIFO kế tiếp.

Khi một trang nhận cơ hội thứ hai, bit tham chiếu của nó được xoá và thời gian đến của nó được đặt lại là thời gian hiện hành. Do đó, một trang được cho cơ hội thứ hai sẽ không được thay thế cho đến khi tất cả trang khác được thay thế (hay được cho cơ hội thứ hai). Ngoài ra, nếu một trang được dùng đủ thường xuyên để giữ bit tham



Hình 6.12. Giải thuật thay thế trang cơ hội thứ hai

chiếu của nó được đặt, nó sẽ không bao giờ bị thay thế.

Một cách để cài đặt giải thuật cơ hội thứ hai như một hàng đợi vòng. Một con trỏ hiển thị trang nào được thay thế tiếp theo. Khi một khung được yêu cầu, con trỏ tăng cho tới khi nó tìm được trang với bit tham chiếu 0. Khi nó tăng, nó xoá các bit tham chiếu (hình 6.12). Một khi trang nạn nhân được tìm thấy, trang được thay thế và trang mới được chèn vào hàng đợi vòng trong vị trí đó. Chú ý rằng, trong trường hợp xấu nhất khi tất cả bit được đặt, con trỏ xoay vòng suốt toàn hàng đợi, cho mỗi trang một cơ hội thứ hai. Thay thế cơ hội thứ hai trở thành thay thế FIFO nếu tất cả bit được đặt.

III.4.3. Giải thuật cơ hội thứ hai nâng cao

Chúng ta có thể cải tiến giải thuật cơ hội thứ hai bằng cách xem xét cả hai bit tham chiếu và sửa đổi như một cặp được xếp thứ tự. Với hai bit này, chúng ta có 4 trường hợp có thể:

1) (0,0) không được dùng mới đây và không được sửa đổi-là trang tốt nhất để thay thế.

2) (0,1) không được dùng mới đây nhưng được sửa đổi-không thật tốt vì trang cần được viết ra trước khi thay thế.

3) (1,0) được dùng mới đây nhưng không được sửa đổi-nó có thể sẽ nhanh chóng được dùng lại.

4) (1,1) được dùng mới đây và được sửa đổi-trang có thể sẽ nhanh chóng được dùng lại và trang sẽ cần được viết ra đĩa trước khi nó có thể được thay thế.

Khi thay thế trang được yêu cầu, mỗi trang ở một trong bốn trường hợp.Chúng ta dùng cùng một cơ chế như giải thuật đồng hồ, nhưng thay vì xem xét trang chúng ta đang trỏ tới có bit tham chiếu được đặt tới 1 hay không, chúng ta xem xét trường hợp mà trang đó đang thuộc về. Chúng ta thay thế trang đầu tiên được gặp trong trường hợp thấp nhất không rỗng. Có thể chúng ta phải quét hàng đợi vòng nhiều lần trước khi chúng ta tìm một trang được thay thế.

Giải thuật này được dùng trong cơ chế quản lý bộ nhớ ảo của Macintosh. Sự khác nhau chủ yếu giữa giải thuật này và giải thuật đồng hồ đơn giản hơn là chúng ta cho tham chiếu tới các trang đó mà chúng được sửa đổi để cắt giảm số lượng nhập/xuất được yêu cầu.

III.4.4. Thay thế trang dựa trên cơ sở đếm

Có nhiều giải thuật khác có thể được dùng để thay thế trang. Thí dụ, chúng ta có thể giữ bộ đếm số lần tham chiếu đối với mỗi trang và phát triển hai cơ chế sau:

- **Giải thuật thay thế trang được dùng ít thường xuyên nhất** (the least frequently used (LFU) page-replacement algorithm) yêu cầu trang với số đếm nhỏ nhất được thay thế. Lý do cho sự chọn lựa này là trang được dùng nên có bộ đếm tham chiếu lớn. Giải thuật này gặp phải trường hợp: trang được dùng nhiều trong tiến trình khởi tạo nhưng không bao giờ được dùng lại. Vì nó được dùng nhiều nên nó có bộ đếm lớn và vẫn ở trong bộ nhớ mặc dù nó không còn cần nữa. Một giải pháp là dịch bộ đếm sang phải 1 bit tại khoảng thời gian đều đặn, hình thành một bộ đếm sử dụng trung bình giảm theo hàm mũ.

- **Giải thuật thay thế trang được dùng thường xuyên nhất** (the most frequently used (MFU) page-replacement algorithm) thay thế trang có giá trị

đếm lớn nhất, nghĩa là trang được sử dụng nhiều nhất.

IV. Cấp phát khung trang

Chúng ta cấp phát lượng bộ nhớ trống cố định giữa các tiến trình khác nhau như thế nào? Nếu chúng ta có 93 khung trang trống và 2 tiến trình, bao nhiêu khung trang mỗi tiến trình sẽ nhận?

Trường hợp đơn giản nhất của bộ nhớ ảo là hệ thống đơn nhiệm. Xét một hệ thống đơn nhiệm với 128 KB bộ nhớ được hình thành từ các trang có kích thước 1 KB. Do đó, có 128 khung trang. Hệ điều hành có thể lấy 35 KB, còn lại 93 khung trang cho tiến trình người dùng. Dưới thuần phân trang yêu cầu, tất cả 93 khung trang đầu tiên được đặt vào danh sách khung trống. Khi một tiến trình người dùng bắt đầu thực thi, nó sinh ra một chuỗi lỗi trang. Những lỗi trang 93 đầu tiên nhận những khung trống từ danh sách khung trống. Khi danh sách khung trống hết, một giải thuật thay thế trang được dùng để chọn một trong 93 trang đang ở trong bộ nhớ để thay thế với trang thứ 94, ...Khi một tiến trình kết thúc, khung trang 93 một lần nữa được thay thế trên danh sách khung trang trống.

Có nhiều thay đổi trên chiến lược đơn giản này. Chúng ta có thể yêu cầu hệ điều hành cấp phát tất cả vùng đệm của nó và không gian bảng từ danh sách khung trống. Khi không gian này không được dùng bởi hệ điều hành, nó có thể được dùng để hỗ trợ phân trang người dùng. Chúng ta có thể cố gắng giữ 3 khung trang trống được dự trữ trên danh sách khung trang trống tại tất cả thời điểm. Do đó, khi lỗi trang xảy ra có một khung trống sẵn có đối với trang. Trong khi hoán chuyển trang xảy ra, một thay thế có thể được chọn, sau đó trang được viết tới đĩa khi tiến trình người dùng tiếp tục thực thi.

Một thay đổi khác cũng có thể thực hiện trên chiến lược cơ bản là tiến trình người dùng được cấp phát bất cứ khung trang nào trống.

Một vấn đề khác phát sinh khi phân trang yêu cầu được kết hợp với đa chương. Đa chương đặt hai hay nhiều tiến trình trong bộ nhớ tại cùng một thời điểm.

IV.1 Số khung trang tối thiểu

Những chiến lược cấp phát khung trang bị ràng buộc trong nhiều cách khác nhau. Chúng ta không thể cấp phát nhiều hơn toàn bộ số khung trang sẵn có (nếu không có chia sẻ trang). Chúng ta cũng cấp phát ít nhất số khung trang tối thiểu. Chú ý, khi số khung trang được cấp phát tới mỗi tiến trình giảm, tỉ lệ lỗi trang tăng, giảm việc thực thi tiến trình.

Ngoài ra, năng lực thực hiện việc cấp phát ngoài mong muốn chỉ có một vài khung trang, có số khung trang tối thiểu phải được cấp phát. Số lượng tối thiểu. Số tối thiểu này được qui định bởi kiến trúc máy tính. Nhớ rằng, khi lỗi trang xảy ra trước khi chỉ thị thực thi hoàn thành, chỉ thị phải bắt đầu lại. Do đó, chúng ta phải có đủ khung trang để giữ tất cả trang khác nhau mà bất cứ chỉ thị đơn có thể tham chiếu.

Thí dụ, xét một máy trong đó tất cả chỉ thị tham chiếu bộ nhớ chỉ có một địa chỉ bộ nhớ. Do đó, chúng ta cần ít nhất một khung trang cho chỉ thị và một khung trang cho tham chiếu bộ nhớ. Ngoài ra, nếu định địa chỉ gián tiếp cấp 1 được phép (thí dụ, một chỉ thị load trên trang 16 có thể tham chiếu tới một địa chỉ bộ nhớ trên trang 0, mà nó tham chiếu gián tiếp tới trang 23), thì phân trang yêu cầu ít nhất 3 khung trên tiến trình. Điều gì có thể xảy ra nếu một tiến trình chỉ có hai khung trang.

IV.2. Các giải thuật cấp phát trang

Có hai tiếp cận:

1. Cấp phát cố định

• Cấp phát công bằng: nếu có m khung trang và n tiến trình, mỗi tiến trình được cấp m/n khung trang

• Cấp phát theo tỉ lệ: dựa vào kích thước của tiến trình để cấp phát số khung trang:

i. Gọi s_i = kích thước của bộ nhớ ảo cho tiến trình p_i

ii. $S = \sum s_i$

iii. m = tổng số khung trang có thể sử dụng

iv. Cấp phát a_i khung trang tới tiến trình p_i: $a_i = (s_i / S)\ m$

2. Cấp phát theo độ ưu tiên

Sử dụng ý tưởng cấp phát theo tỷ lệ, nhưng lượng khung trang cấp cho tiến trình phụ thuộc vào độ ưu tiên của tiến trình hơn là phụ thuộc kích thước tiến trình. Nếu tiến trình p_i phát sinh lỗi trang, chọn một trong các khung trang của nó để thay thế, hoặc chọn một khung trang của tiến trình khác với độ ưu tiên thấp hơn để thay thế.

• Thay thế trang toàn cục hay cục bộ. Có thể phân các thuật toán thay thế trang thành hai lớp chính:

i. Thay thế toàn cục: khi lỗi trang xảy ra với một tiến trình, chọn trang "nạn nhân" từ tập tất cả các khung trang trong hệ thống, bất kể khung trang đó đang được cấp phát cho một tiến trình khác.

ii. Thay thế cục bộ: yêu cầu chỉ được chọn trang thay thế trong tập các khung trang được cấp cho tiến trình phát sinh lỗi trang

Một khuyết điểm của giải thuật thay thế toàn cục là các tiến trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình. Vì thế, tuy giải thuật thay thế toàn cục nhìn chung cho phép hệ thống có nhiều khả năng xử lý hơn, nhưng nó có thể đẩ hệ thống đến tình trạng trì trệ toàn bộ hệ thống (thrashing).

V. Trì trệ toàn hệ thống

Nếu một tiến trình không có đủ các khung trang để chứa những trang cần thiết cho xử lý thì nó sẽ thường xuyên phát sinh lỗi trang và vì thế phải dùng đến rất nhiều thời gian sử dụng CPU để thực hiện thay thế trang. Một hoạt động phân trang như thế được gọi là sự trì trệ (thrashing). Một tiến trình lâm vào trạng thái trì trệ nếu nó sử dụng nhiều thời gian để thay thế hơn là để xử lý. Hiện tượng này ảnh hưởng nghiêm trọng đến hoạt động hệ thống, xét tình huống sau:

1) Hệ điều hành giám sát việc sử dụng CPU

2) Nếu hiệu suất sử dụng CPU quá thấp, hệ điều hành sẽ nâng mức độ đa chương bằng cách đưa thêm một tiến trình mới vào hệ thống.

3) Hệ thống có thể sử dụng giải thuật thay thế toàn cục để chọn các trang nạn nhân thuộc một tiến trình bất kỳ để có chỗ nạp tiến trình mới, có thể sẽ thay thế cả các trang của tiến trình đang xử lý hiện hành.

4) Khi có nhiều tiến trình trong hệ thống hơn, thì một tiến trình sẽ được cấp ít khung trang hơn và do đó phát sinh nhiều lỗi trang hơn.

5) Khi các tiến trình phát sinh nhiều lỗi trang, chúng phải trải qua nhiều thời gian chờ các thao tác thay thế trang hoàn tất, lúc đó hiệu suất sử dụng CPU lại giảm.

6) Hệ điều hành lại quay trở lại bước 1.

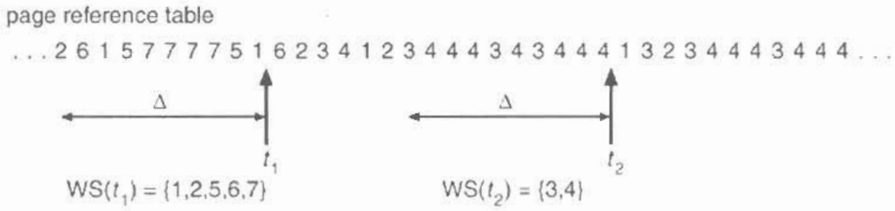
Theo kịch bản trên đây, hệ thống sẽ lâm vào tình trạng luẩn quẩn của việc giải phóng các trang để cấp phát thêm khung trang cho một tiến trình, và các tiến trình khác lại thiếu khung trang..và các tiến trình không thể tiếp tục xử lý. Đây chính là tình trạng trì trệ toàn bộ hệ thống. Khi tình trạng trì trệ này xảy ra, hệ thống gần như mất khả năng xử lý, tốc độ phát sinh lỗi trang tăng rất cao không công việc nào có thể kết thúc vì tất cả tiến trình đều bận rộn với việc phân trang. Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho tiến trình đủ các khung trang cần thiết để hoạt động. Vấn đề cần giải quyết là làm sao biết được quá trình cần bao nhiêu trang?

V.1 Mô hình cục bộ

Theo lý thuyết cục bộ thì khi một tiến trình xử lý nó có khuynh hướng di chuyển từ nhóm trang cục bộ này đến nhóm trang cục bộ khác. Một nhóm trang cục bộ là một tập các trang đang được tiến trình dùng đến trong một khoảng thời gian. Một chương trình thường bao gồm nhiều nhóm trang cục bộ khác nhau và chúng có thể giao nhau.

V.2 Mô hình tập làm việc

Mô hình tập làm việc (working set model) này dựa trên cơ sở lý thuyết cục bộ. Mô hình này sử dụng tham số Δ để định nghĩa cửa sổ cho tập làm việc. Giả sử, khảo sát Δ đơn vị thời gian (lần truy xuất trang) cuối cùng, tập các trang được tiến trình truy xuất đến trong Δ lần truy cập cuối cùng được gọi là tập làm việc của tiến trình tại thời



Hình 6.13. Mô hình tập làm việc

điểm hiện tại. Nếu một trang đang được tiến trình truy xuất tới, nó sẽ nằm trong tập làm việc nếu nó không sử dụng nữa, nó sẽ bị loại khỏi tập làm việc của tiến trình sau Δ đơn vị thời gian kể từ lần truy xuất cuối cùng đến nó. Như vậy, tập làm việc chính là sự xấp xỉ của khái niệm nhóm trang cục bộ.

Thuộc tính rất quan trọng của tập làm việc là kích thước của nó. Nếu tính toán kích thước tập làm việc WSS_i, cho mỗi tiến trình trong hệ thống thì có thể xem:

$$D = \sum WSS_i$$

Với D là tổng số khung trang yêu cầu cho toàn hệ thống. Mỗi tiến trình sử dụng các trang trong tập làm việc của nó, nghĩa là tiến trình i yêu cầu WSS_i khung trang. Nếu tổng số trang yêu cầu vượt quá tổng số trang có thể sử dụng trong hệ thống (D >m), thì sẽ xảy ra tình trạng trì trệ toàn bộ.

Dùng mô hình tập làm việc là đơn giản. Hệ điều hành kiểm soát tập làm việc

của mỗi tiến trình và cấp phát cho tiến trình tối thiểu các khung trang để chứa đủ tập làm việc của nó. Nếu có đủ khung trang bổ sung thì tiến trình khác có thể được khởi tạo. Nếu tổng kích thước tập làm việc gia tăng vượt quá tổng số khung sẵn có, hệ điều hành chọn một tiến trình để tạm dừng. Những trang của tiến trình này được viết ra đĩa và các khung trang của nó được cấp phát lại cho tiến trình khác. Tiến trình được tạm dừng có thể khởi động lại sau đó.

Chiến lược tập làm việc ngăn chặn sự trì trệ trong khi giữ cấp độ đa chương cao nhất có thể. Do đó, nó tối ưu việc sử dụng CPU. Khó khăn với mô hình tập làm việc này là giữ vết của tập làm việc. Cửa sổ tập làm việc là một cửa sổ di chuyển. Tại mỗi tham chiếu bộ nhớ, một tham chiếu mới xuất hiện khi một tham chiếu trước đó kết thúc và tham chiếu cũ nhất trở thành điểm kết thúc khác. Một trang ở trong tập làm việc nếu nó được tham chiếu bất cứ nơi nào trong cửa sổ tập làm việc. Chúng ta có thể xem mô hình tập làm việc gần xấp xỉ với ngắt đồng hồ sau từng chu kỳ cố định và bit tham chiếu.

V.3 Tần suất lỗi trang

Tần suất lỗi trang rất cao khiến tình trạng trì trệ hệ thống xảy ra. Khi tần suất lỗi trang quá cao, tiến trình cần thêm một số khung trang. Ngược lại, khi tần suất quá thấp, tiến trình có thể sở hữu nhiều khung trang hơn mức cần thiết. Có thể thiết lập một giá trị cận trên và cận dưới cho tần suất xảy ra lỗi trang và trực tiếp ước lượng và kiểm soát tần suất lỗi trang để ngăn chặn tình trạng trì trệ xảy ra:

- Nếu tần suất lỗi trang vượt quá cận trên, cấp cho tiến trình thêm một khung trang
- Khi tần suất lỗi trang thấp hơn cận dưới, thu hồi bớt một khung trang từ tiến trình.

Với chiến lược tập làm việc, chúng ta có thể có phải tạm dừng một tiến trình. Nếu tỉ lệ lỗi trang tăng và không có trang nào trống, chúng ta phải chọn một số quá trình và tạm dừng nó. Sau đó, những khung trang được giải phóng sẽ được phân phối lại cho các tiến trình với tỉ lệ lỗi trang cao.

VI. Các vấn đề khác

VI.1. Kích thước trang

Kích thước trang thông thường được xác định bởi phần cứng. Không có sự chọn lựa lý tưởng cho kích thước trang:

- Kích thước trang càng lớn thì kích thước bảng trang càng giảm
- Kích thước trang càng nhỏ thì cho phép tổ chức nhóm trang cục bộ tốt hơn và giảm sự phân mảnh trong.
- Thời gian nhập xuất nhỏ khi kích thước trang lớn
- Kích thước trang nhỏ thì có thể giảm số lượng thao tác nhập xuất cần thiết vì có thể xác định các nhóm trang cục bộ chính xác hơn
- Kích thước trang lớn sẽ giảm tần xuất lỗi trang

Đa số các hệ thống chọn kích thước trang là 4 KB.

VI.2. Cấu trúc chương trình

Về nguyên tắc, kỹ thuật phân trang theo yêu cầu được thiết kế nhằm giúp người dùng khỏi bận tâm đến việc sử dụng bộ nhớ một cách hiệu quả. Tuy nhiên, nếu hiểu rõ tổ chức bộ nhớ trong kỹ thuật phân trang, lập trình viên có thể giúp cho hoạt động của hệ thống tốt hơn với chương trình được xây dựng phù hợp.

Thí dụ, giả sử 1 trang có kích thước 128 bytes, một chương trình khởi tạo và gán giá trị mảng có kích thước 128x128 như sau:

```
Var A: array[1..128] of array [1..128] of byte;
For i:= 1 to 128 do
        For j:=1 to 128 do
                A[i][j]:=0;
```

Trong Pascal, C, PL/I, mảng trên đây được lưu trữ theo thứ tự dòng, mỗi dòng mảng chiếm một trang bộ nhớ, do đó tổng số lỗi trang phát sinh sẽ là 128. Trong Fortran, mảng trên đây lại được lưu trữ theo thứ tự cột, do đó tổng số lỗi trang phát sinh sẽ là 128x128 = 1638.

VI.3. Neo các trang trong bộ nhớ chính

Khi áp dụng kỹ thuật phân trang đôi lúc có nhu cầu "neo" trong bộ nhớ chính một số trang quan trọng hoặc thường được sử dụng hoặc không thể chuyển ra bộ nhớ phụ để bảo toàn dữ liệu.

Khi đó sử dụng thêm một bit khoá gán tương ứng cho từng khung trang. Một khung trang có bit khoá được đặt sẽ không bị chọn để thay thế.

VII. Ghi nhớ.

Mong muốn có thể thực thi một tiến trình có không gian địa chỉ lô gic lớn hơn không gian địa chỉ vật lý sẵn có. Người lập trình có thể làm một tiến trình như thế có thể thực thi bằng cách cấu trúc lại nó dùng cơ chế phủ lắp, nhưng thực hiện điều này thường là một tác vụ lập trình khó. Bộ nhớ ảo là một kỹ thuật cho phép không gian địa chỉ lô gic được ánh xạ vào bộ nhớ vật lý nhỏ hơn. Bộ nhớ ảo cho phép những quá trình cực lớn được chạy và cũng cho phép cấp độ đa chương được gia tăng, tăng khả năng sử dụng CPU. Ngoài ra, nó giải phóng người lập trình ứng dụng từ việc lo lắng khả năng sẵn có của bộ nhớ.

Thuần phân trang theo yêu cầu mang vào một trang cho tới khi trang đó được tham chiếu. Tham chiếu đầu tiên gây ra lỗi trang tới hệ điều hành. Hệ điều hành xem xét bảng trang bên trong để xác định nơi trang được định vị trên vùng bộ nhớ phụ.

Bảng trang được cập nhật để phản ánh sự thay đổi này, cho phép một tiến trình chạy mặc dù toàn bộ hình ảnh bộ nhớ của nó không ở trong bộ nhớ chính. Khi tỉ lệ lỗi trang tương đối thấp, năng lực có thể chấp nhận.

Chúng ta có thể dùng phân trang theo yêu cầu để giảm số khung trang được cấp phát tới tiến trình. Sắp xếp này có thể tăng cấp độ đa chương (cho phép nhiều quá trình sẵn sàng thực thi tại một thời điểm). Nó cũng cho phép các tiến trình được thực thi mặc dù yêu cầu bộ nhớ vượt quá toàn bộ bộ nhớ vật lý sẵn có. Những tiến trình như thế chạy trong bộ nhớ ảo.

Nếu tổng số yêu cầu bộ nhớ vượt quá bộ nhớ vật lý, thì nó cần thay thế trang từ bộ nhớ tới các khung trang trống cho những trang mới. Những giải thuật thay thế trang khác nhau được dùng. Thay thế trang FIFO là dễ dàng đối với chương trình nhưng gặp

phải lỗi Belady. Thay thế trang tối ưu yêu cầu kiến thức tương lai. Thay thế LRU là xấp xỉ tối ưu nhưng nó rất khó cài đặt. Hầu hết các giải thuật thay thế trang như giải thuật cơ hội thứ hai là xấp xỉ thay thế LRU.

Ngoài ra đối với giải thuật thay thế trang, chính sách cấp phát khung trang được yêu cầu. Cấp phát có thể cố định, đề nghị thay thế trang cục bộ, hay động, đề nghị thay thế toàn cục. Mô hình tập làm việc cho rằng các tiến trình thực thi trong các vị trí. Tập làm việc là tập các trang trong các vị trí hiện hành. Theo đó, mỗi tiến trình nên được cấp phát đủ các khung cho tập làm việc hiện hành của nó.

Nếu một tiến trình không có đủ bộ nhớ cho tập làm việc của nó, nó sẽ bị trì trệ. Cung cấp đủ khung cho mỗi tiến trình để tránh trì trệ có thể yêu cầu tiến trình hoán chuyển và định thời.

Ngoài ra, để yêu cầu chúng ta giải quyết các vấn đề chính của thay thế trang và cấp phát khung trang, thiết kế hợp lý hệ thống phân trang yêu cầu chúng ta xem xét kích thước trang, nhập/xuất, khoá, phân lại trang, tạo tiến trình, cấu trúc chương trình, sự trì trệ,.. Bộ nhớ ảo có thể được xem như một cấp của cơ chế phân cấp trong các cấp lưu trữ trong hệ thống máy tính. Mỗi cấp có thời gian truy xuất, kích thước và tham số chi phí của chính nó.